Titre: Title:	Généralisation du modèle delta-lognormal pour l'étude des mouvements en trois dimensions
Auteur: Author:	Nicolas Leduc
Date:	2001
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Leduc, N. (2001). Généralisation du modèle delta-lognormal pour l'étude des mouvements en trois dimensions [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8768/

UNIVERSITÉ DE MONTRÉAL

GÉNÉRALISATION DU MODÈLE DELTA-LOGNORMAL POUR L'ÉTUDE DES MOUVEMENTS EN TROIS DIMENSIONS

NICOLAS LEDUC
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
MARS 2001

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

GÉNÉRALISATION DU MODÈLE DELTA-LOGNORMAL POUR L'ÉTUDE DES MOUVEMENTS EN TROIS DIMENSIONS

présenté par: LEDUC Nicolas

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. SAWAN Mohamad, Ph.D., président

M. PLAMONDON Réjean, Ph.D., membre et directeur de recherche

M. AÏSSAOUI Rachid, Ph.D., membre

À Jean-Claude. Louise, Mathieu et Olivier

Remerciements

Je tiens premièrement à remercier Réjean Plamondon, mon directeur de recherche, qui m'a soutenu tout au long de mon projet de maîtrise et n'a jamais cessé de m'encourager. Sans lui ce travail n'aurait jamais vu le jour. Je voudrais aussi remercier Wacef Guerfali qui, du temps où il était au laboratoire Scribens, m'a aidé et guidé tout au long de mon cheminement.

J'aimerais également remercier les membres du jury qui ont accepté d'évaluer le travail réalisé et la qualité scientifique de celui-ci, ainsi que le Département de Génie Électrique de l'École Polytechnique de Montréal pour m'avoir fourni des équipements nécessaires à la réalisation de ce projet ainsi qu'un cadre de travail agréable et stimulant. Je tiens aussi à remercier le Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG) pour m'avoir décerné une bourse d'études sans laquelle il m'aurait été impossible de compléter ce projet.

Je m'en voudrais de passer sous silence l'aide que les chercheurs du laboratoire Scribens m'ont fournie. Je tiens à remercier Hong Trang Nguyen, Frédérick Jubinville et Laurant Charlin qui se sont prêtés patiemment à mes expérimentations ainsi que Jean-Guy Deschênes et Olivier Adam sans qui les photos présentées dans ce document n'aurait pas été possibles. Je voudrais aussi remercier Suzanne Morgan pour m'avoir aidé à traduire et à corriger mon résumé.

Enfin, je voudrais remercier ma famille qui n'a jamais cessé de croire en moi et de m'encourager à continuer. Sans leur soutien constant et un cadre familial exceptionnel, je crois qu'il m'aurait été impossible d'entreprendre un tel projet.

Résumé

Chaque jour des chercheurs travaillent afin de mieux comprendre le fonctionnement du corps humain et essaient d'imiter ses comportements. Dans cette perspective, depuis une centaine d'années, différents modèles ont été proposés pour reproduire les mouvements des humains. On s'est intéressé à la reconstruction des profils de vitesse ainsi que des trajectoires observées.

Plamondon a proposé, en 1993, le modèle delta-lognormal dans le cadre d'une théorie cinématique de génération de mouvements. Ce modèle offre une performance de reconstruction des profils de vitesse dépassant tout ce qui avait été proposé jusqu'à ce jour. Quelques années plus tard, il fut modifié pour être en mesure d'analyser des traces en deux dimensions. Cette extension permit d'extraire des paramètres et de reconstruire, entre autres, des signatures manuscrites. Pour continuer ces travaux, il fallait généraliser le modèle delta-lognormal en trois dimensions et c'est l'objectif de ce mémoire.

Pour atteindre cet objectif, nous commençons par faire une revue bibliographique des modèles existants en une, deux et trois dimensions. Les modèles semblables sont regroupés. Certains modèles sont purement mathématiques alors que d'autres sont fondés sur la stratégie utilisée pour générer des mouvements. Le modèle delta-lognormal, étant en mesure de reconstruire seulement les profils de vitesse, est placé dans la catégorie des modèles à une dimension.

Nous présentons ensuite le modèle delta-lognormal en détails. Ce modèle est basé sur la définition d'un geste simple qui résulte d'une synergie impliquant l'activation de deux systèmes neuromusculaires, l'un agoniste et l'autre antagoniste. Les fondements, sur

lesquels le modèle est basé, sont expliqués et des résultats obtenus lors de la reconstruction de mouvements simples sont montrés. Ce modèle de base comporte sept paramètres.

L'extension du modèle à deux dimensions a nécessité l'ajout de deux paramètres au modèle de base. Ces paramètres ont été nécessaires pour spécifier l'orientation et la courbure du tracé dans le plan. Un extracteur automatique de paramètre a été programmé dans le but d'extraire les paramètres sur des signatures ou du texte manuscrit. La performance du modèle en deux dimensions est excellente. Le modèle delta-lognormal a cependant un problème, les solutions ne sont pas uniques ce qui rend l'extraction des paramètres difficile. Ceci est vrai autant pour le modèle en une dimension que pour celui en deux dimensions.

Dans ce mémoire, nous proposons des extensions à trois dimensions pour les modèles publiés dans la littérature et, ce faisant, nous en venons à la conclusion que le modèle delta-lognormal est le plus apte à reconstruire des mouvements en trois dimensions. L'utilisation des équations de Frenet a été une avenue explorée, mais elle n'a pas été retenue, car elle ne correspondait pas à ce que nous observions pour des mouvements simples. Comme un mouvement simple se produit habituellement dans un plan, nous avons ajouté deux paramètres au modèle précédent, pour un total de onze. Ces nouveaux paramètres correspondent à l'orientation du plan contenant le geste. Des simulations de gestes simples et complexes nous permettent de croire que le modèle sera en mesure de reconstruire des mouvements réels.

Nous avons fait l'acquisition de mouvements simples à l'aide de l'appareil "A Flock of Birds" de la compagnie Ascension. Cet appareil permet de connaître la position, en trois dimensions, de capteurs en analysant les courants induits dans des bobines, par un champ magnétique généré. La fréquence d'échantillonnage que nous avons utilisée est de 97,9 Hz. Les fichiers qui contiennent les points issus de l'acquisition ont été transformés pour être

en mesure d'être lus à l'aide d'un programme d'extraction de paramètres. Ce dernier filtre les données à l'aide d'un filtre passe bas ayant une fréquence de coupure de 16 Hz avant de faire l'extraction proprement dite. Il optimise les paramètres à l'aide d'une méthode de régression non linéaire. Un programme a aussi été conçu, dans le cadre de ce projet, dans le but de visualiser les trajectoires ainsi que les profils de vitesse obtenus. Celui-ci permet aussi de modifier les paramètres du modèle delta-lognormal en trois dimensions et de voir le résultat immédiatement à l'écran. Il utilise les librairies Win32 et OpenGL.

Les résultats obtenus sur les membres supérieurs permettent d'affirmer que le modèle s'applique pour l'analyse de ceux-ci. Par contre, les profils de vitesse obtenus pour les membres inférieurs sont difficiles à décrire à l'aide du modèle. Il est difficile de savoir, à ce jour, s'il s'agit d'un fait réel ou d'un artefact causé par un manque d'entraînement des sujets en regard du protocole expérimental utilisé. Des tests ont été faits sur quarante-huit mouvements simples exécutés par deux sujets. Les conditions expérimentales n'étaient pas optimales à cause de l'endroit où a été installé l'appareil d'acquisition. Malgré cela, l'analyse de la courbure nous permet de penser qu'il sera possible de différencier les différents membres.

Le modèle delta-lognormal en trois dimensions a plusieurs applications possibles, mais celles-ci ne sont pas présentées dans ce mémoire, car elles dépassent le cadre du projet. Cependant, le modèle pourrait permettre de déceler des maladies neuro-motrices dégénératives comme la maladie de Parkinson et de suivre leur évolution. De plus, il pourrait être utilisé pour l'élaboration de stimulateurs intra-musculaires pour permettre de redonner de la motricité aux personnes handicapées. Des applications sont aussi possibles dans le domaine de l'animation par ordinateur ou de la sécurité informatique.

Abstract

Every day many researchers work to better understand the operation of the human body and try to imitate its behaviour. For more than a hundred years, many models have been proposed to reproduce human movements. Some researchers were interested in the modeling of the velocity profiles and trajectories observed.

Plamondon proposed, in 1993, a new model in the framework of a kinematic theory of movement generation, the delta-lognormal model. Since then, no model has been found which outperforms this one for the reconstruction of the velocity profiles observed. To be able to analyze two dimensional human movements, the model was modified a few years later. This extended model is able to reconstruct handwritten signature if the right parameters are extracted and used. To continue this work, the model has to be generalized to the third dimension and this is the object of the persent work.

To achieve our goal, we start by doing a bibliographical review of the models, in one, two and three dimensions, as found in the literature. We gather similar models. Some models have their origins from the strategy used to generate movements and others are purely mathematical. Because the delta-lognormal model is only able to reconstruct velocity profiles, it is put put in the one-dimensional category.

Next, we present the details of the delta-lognormal model. This model is based on the definition of a simple gesture. The latter could be considered as a synergy generated by the synchronous activation of two neuromuscular systems, one agonist and the other antagonist. The basis of this model are explained and the results obtained from the reconstruction of simple gestures are shown. Seven parameters are used in this basic model.

To be able to make the delta-lognormal model work in two dimensions, two parameters were added to the basic model. These parameters were needed to specify the orientation and the curvature of the two dimensional stroke. The parameters were extracted from signature or handwritten text by an automatic extractor programmed for this purpose. We obtained excellent results with the two dimensional model, but multiple solutions are possible so the extraction of the parameters is difficult. This is true for the model in one and two dimensions.

In this document, we propose three dimensional extensions to the models found in the literature. We conclude, from these propositions, that the delta-lognormal model is the one that can give the best performance for the reconstruction of three dimensional strokes. We tested a model using the Frenet formula, but this approach was not retained because it is not representative of the simple movements observed. Since a simple movement is excuted in a plane, we added two more parameters to the two dimensional model for a total of eleven. The plane containing the gesture is represented by these two new parameters. Some simulations for simple and complex movements demonstrate that the model will be able to reconstruct real human movements.

We use the system "A Flock of Birds" from the compagny Ascension to acquire simple movements. This system is able to know the position, in space, of the sensors by analyzing the currents induced in coils by the magnetic fields generated. The sampling rate we used is 97.9 Hz. The files containing the points acquired were transformed so that we could read them by the parameter extraction software. The latter filters the data with a low pass filter that has a cut-off frequency of 16 Hz before extraction. The program use a non-linear regression to optimize the data. We developed a software, within the framework of this project, to visualize the three dimensional strokes and the velocity profiles obtained. With this software we can also modify the model's parameters and see the results on the

reconstruction of the three dimensional strokes immediately. It uses the Win32 and OpenGL libraries.

The model has been applied to analyze the upper limbs and the results obtained validate the model for those movements. However, the velocity profiles for the lower limbs are not totally explained by the model. It is difficult, to date, to know if it is a fact or an artefact caused by a lack of training for the subjects because of the experimental protocol used. Two subjects executed forty-eight simple gestures each. The room where the machine was installed was not optimal for this type of experiment. Preliminary analysis of the curvature of the movements indicates to us that it will be possible to differentiate gesture, in spite of the poor experimental conditions.

The three dimensional delta-lognormal model has many possible applications. These one were not presented in this document because they exceed the framework of this project. However, the model may be able to detect neuro-motor diseases like Parkinson's disease, and follow their evolution. It could also be used to help develop intramuscular implants to give back motricity to handicapped people. Some applications are possible in the field of computer animation or for data security.

Table des matières

Liste des tableaux

Liste des figures

Chapitre 1
Introduction et recherche bibliographique

1.1 Introduction

Au cours du dernier siècle, beaucoup de chercheurs se sont intéressés à la modélisation du corps humain. Chacun d'eux a des buts et des objectifs propres. Certains veulent comprendre le fonctionnement des organes du point de vue microscopique et d'autres du point de vue macroscopique. Toutes ces recherches ont cependant un but commun : être en mesure de mieux comprendre les mécanismes du corps humain afin d'utiliser ces connaissances pour différentes applications.

Au laboratoire Scribens de l'École Polytechnique de Montréal, nous sommes intéressés à la modélisation des gestes. Il y a quelques années, Plamondon (1995a,b) a proposé un modèle pour expliquer les courbes de vitesse observées lors de mouvements rapides. Il l'a appliqué tout d'abord sur des gestes simples en une dimension puis à deux dimensions. Son modèle, nommé modèle delta-lognormal, permet la reconstruction des courbes de vitesse et des traces observées pour des gestes aussi complexes que des signatures. Cette approche conduit à des erreurs de reconstruction moindres que celles obtenues en utilisant plusieurs autres modèles (Alimi, 1995).

Comme le modèle est très performant en une et deux dimensions, nous avons pensé le généraliser pour l'étude des mouvements en trois dimensions. En effet, si nous avons une excellente performance en deux dimensions, il est probable que le modèle en trois

dimensions aura une performance environ égale ou supérieure à celle des modèles existants. Cette généralisation constitue le sujet du présent mémoire, soit la modélisation des mouvements humains en trois dimensions à l'aide du modèle delta-lognormal. Plus particulièrement, nous proposons un modèle qui se base sur celui en deux dimensions et nous le validons en l'appliquant à l'étude de gestes tridimensionnels simples.

La généralisation du modèle n'est pas un problème trivial. En effet, il ne s'agira pas seulement d'appliquer des équations, mais il faudra, de plus, bien comprendre leurs implications au niveau physiologique, ce qui rend la tâche complexe. Autrement les équations de Frenet pourraient s'appliquer directement. Nous avons donc fait attention lors de l'élaboration du modèle de ne jamais perdre de vue les implications au niveau physiologique.

Ce premier chapitre est divisé en trois sections. Dans la première nous verrons les motivations qui nous ont amené à développer un modèle général en trois dimensions. La seconde section présente les principaux modèles qui ont été exposés dans la littérature à ce jour. Ceux-ci sont regroupés en trois catégories, soit ceux en une deux et trois dimensions. La dernière section présente le travail qui a été réalisé au cours de cette recherche ainsi que le plan du mémoire.

1.2 Motivation du choix du sujet

Le choix de ce projet est motivé par le fait que les modèles en trois dimensions sont peu nombreux et généralement spécialisés. Comme la technologie d'acquisition en trois dimensions est jeune, peu de chercheurs ont des appareils d'acquisition performants à leur disposition. Par contre, les outils pour faire l'acquisition de traces en deux dimensions sont beaucoup plus répandus, ce qui a comme conséquence de permettre à beaucoup de

chercheurs de proposer des modèles à partir de leurs expérimentations. Ce phénomène fait que les modèles pour analyser et étudier les mouvements dans l'espace sont peu nombreux.

Les modèles proposés sont aussi généralement spécialisés. En effet, les chercheurs qui analysent les mouvements le font habituellement dans un but précis. Prenons, par exemple, un groupe de recherche qui travaille sur le mouvement de marche. Ce groupe s'intéresse avant tout aux modèles qui ne s'appliquent qu'aux jambes lors de la marche. Ces modèles sont donc très spécifiques et ne peuvent généralement pas être utilisés pour les autres membres.

Comme nous disposons de l'appareillage nécessaire à l'acquisition des mouvements dans l'espace, nous avons décidé de travailler sur un modèle plus général. Nous voulons donc tenter de combler une lacune des modèles existants, soit que ceux-ci ne s'appliquent que pour des mouvements spécifiques. De plus, nous voulons proposer un modèle qui permettra de caractériser éventuellement, à l'aide de différents paramètres, des personnes et des gestes. En effet, il serait intéressant de pouvoir caractériser les gestes et les personnes, car ceci pourrait avoir plusieurs applications, comme l'authentification de personnes ou le dépistage de maladies neuro-motrices. Comme notre modèle pourra probablement s'appliquer à tous les membres et qu'il contiendra des paramètres qui dépendent de la physiologie de la personne, nous croyons qu'il pourrait être possible, à moyen terme, d'identifier des gestes ou des personnes.

Une autre raison qui motive le choix du modèle est sa simplicité et son petit nombre de paramètres, si nous le comparons à d'autres modèles. Nous proposons un modèle qui associe onze paramètres par maximum, ou pic, de vitesse ce qui est peu si nous comparons à d'autres modèles qui utilisent, par exemple, des réseaux de neurones. De plus, nous ne faisons pas intervenir d'équations différentielles, donc nous n'avons pas de système

d'équations différentielles à solutionner. Nous avons donc de bonnes raisons de croire que notre modèle sera probablement plus simple à utiliser et à analyser.

1.3 Modèles antérieurs

Jusqu'à présent nous avons présenté de manière très générale le projet qui consiste en la modélisation des mouvements humains en trois dimensions à l'aide du modèle delta-lognormal. Nous avons aussi discuté des raisons qui motivent notre travail. Nous allons maintenant présenter le travail qui a été fait jusqu'à présent dans le domaine de la modélisation des mouvements humains en décrivant chacun des modèles que nous avons trouvés dans la littérature. Ceux-ci seront classés en trois groupes soit ceux en une, deux et trois dimensions. Ceux-ci seront repris au troisième chapitre dans le but de proposer une généralisation de chacun.

1.3.1 Modèles en une dimension

Comme la littérature est abondante au sujet des modèles en une et deux dimensions, nous allons débuter par ceux-ci. La thèse de Alimi (1995) et l'article de Plamondon et al. (1993) présentent 26 modèles dans le but de les comparer. Pour permettre une comparaison, ils ont écrit l'équation du profil de vitesse pour chaque modèle en fonction de certains paramètres. Nous allons ici utiliser la même notation, car elle nous permettra de regrouper les modèles. Nous allons présenter brièvement chacun de ces modèles en nous inspirant fortement de la présentation faite par Alimi (1995) dans sa thèse.

Le modèle de Denier van der Gon (1962) fut le premier modèle d'écriture. Il propose qu'un geste rapide est généré par l'activation de deux groupes musculaires perpendiculaires. Dooijes (1983) modifie le modèle en y faisant deux ajouts. Il propose

une vitesse horizontale constante et utilise des axes obliques. Ce dernier modèle est décrit par l'équation (1.1) où l'on constate un profil de vitesse asymétrique et discontinu.

$$V_\sigma(t)=\begin{cases}\sqrt{[P_1(1-e^{-P_2(t-P_3)})]^2+[P_4(1-e^{-P_5(t-P_6)})]^2} & T_0\le t\le T_m\\ \sqrt{[P_7e^{-P_8(t-P_9)}]^2+[P_{10}e^{-P_{11}(t-P_{12})}]^2} & T_m\le t\le T_1\\ 0 & ailleurs\end{cases} \quad (1.1)$$

MacDonald (1964) a proposé un modèle asymétrique et discontinu. Son modèle avait pour but d'améliorer les performances du modèle de Denier van der Gan (1962). En utilisant une impulsion de forme trapézoïdale au lieu d'une rectangulaire, il obtient un profil de vitesse décrit par l'équation (1.2).

$$V_\sigma(t)=\begin{cases}\sqrt{[P_1(1-e^{-P_2(t-P_3)})]^2+[P_4(1-e^{-P_5(t-P_6)})]^2} & T_0\le t\le T_2\\ \sqrt{[P_7e^{-P_8(t-P_{19})}+(P_9t-P_{19})]^2+[P_{10}e^{-P_{11}(t-P_{19})}+(P_{12}t-P_{19})]^2} & T_2\le t\le T_3\\ \sqrt{[P_{13}e^{-P_{14}(t-P_{15})}]^2+[P_{16}e^{-P_{17}(t-P_{18})}]^2} & T_3\le t\le T_1\\ 0 & ailleurs\end{cases} \quad (1.2)$$

Après plusieurs analyses Yasuhara (1975) a démontré que les forces ayant des transitions de type exponentielles permettaient une meilleure reconstruction des mouvements d'écriture. Il a donc proposé le modèle (1.3) qui est continu et asymétrique.

$$V_\sigma(t)=\begin{cases}\sqrt{[P_1e^{-P_2(t-P_9)}+P_3e^{-P_4(t-P_9)}]^2+[P_5e^{-P_6(t-P_9)}+P_7e^{-P_8(t-P_9)}]^2} & T_0\le t\le T_1\\ 0 & ailleurs\end{cases} \quad (1.3)$$

En utilisant des impulsions triangulaires pour l'accélération qui servent d'entrée à une série d'intégrateurs, Maarse (1987) a proposé un système purement balistique. Nous obtenons un profil de vitesse asymétrique discontinu décrit par l'équation (1.4).

$$V_o(t)=\begin{cases}\sqrt{[P_1(t-P_2)(t-P_3)]^2+[P_4(t-P_5)(t-P_6)]^2} & T_0\le t\le T_m \\ \sqrt{[P_7(t-P_8)(t-P_9)]^2+[P_{10}(t-P_{11})(t-P_{12})]^2} & T_m\le t\le T_1 \\ 0 & \textit{ailleurs}\end{cases} \qquad (1.4)$$

L'utilisation des oscillateurs harmoniques pour modéliser la réponse des muscles a permis à Eden (1962) et Hollerbach (1981) d'obtenir un profil de vitesse symétrique continu (1.5). Dans ce modèle deux oscillateurs orthogonaux ont été utilisés.

$$V_o(t)=\begin{cases}\sqrt{\{P_1\sin[P_2(t-P_3)]+P_4\}^2+\{P_5\sin[P_6(t-P_7)]\}^2} & T_0\le t\le T_1 \\ 0 & \textit{ailleurs}\end{cases} \qquad (1.5)$$

Pour obtenir un profil asymétrique, Mermelstein et Eden (1964) ont modifié le modèle de Eden (1962) en donnant des paramètres différents pour les phases montantes et descendantes du profil de vitesse. Le problème est que l'équation (1.6) du profil est discontinue.

$$V_o(t)=\begin{cases}\sqrt{\{P_1\sin[P_2(t-P_3)]+P_4\}^2+\{P_5\sin[P_6(t-P_7)]\}^2} & T_0\le t\le T_m \\ \sqrt{\{P_8\sin[P_9(t-P_{10})]+P_{11}\}^2+\{P_{12}\sin[P_{13}(t-P_{14})]\}^2} & T_m\le t\le T_1 \\ 0 & \textit{ailleurs}\end{cases} \qquad (1.6)$$

Plamondon et Lamarche (1986) ont utilisé l'équation de transfert d'un moteur à courant continu pour modéliser les muscles comme des générateurs de vitesse. Nous obtenons l'équation (1.7) qui produit des profils de vitesse asymétriques discontinus. L'impulsion utilisée pour activer le système est indépendante de la direction du mouvement.

$$V_o(t)=\begin{cases} P_1(1-e^{-P_2(t-P_3)}) & T_0 \le t \le T_m \\ P_4 e^{-P_5(t-P_6)} & T_m \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.7}$$

Si nous considérons un mouvement comme une série de mouvements de base et que chacun de ceux-ci est modélisé par un profil de vitesse ayant une forme de cloche symétrique, nous obtenons le modèle de base de Morasso et Mussa-Ivaldi (1982). Pour obtenir ces cloches, ils utilisent la fonction de la spline cubique.

$$V_o(t)=\begin{cases} P_1(t-P_2)(t-P_3) & T_0 \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.8}$$

Si nous utilisons des fonctions cosinus à la place d'utiliser des splines cubiques pour modéliser le profil de vitesse, nous obtenons le modèle de Maarse (1987). Il en a proposé deux versions soit une continue et symétrique (1.9) et l'autre qui est asymétrique et discontinue (1.10).

$$V_o(t)=\begin{cases} P_1\{1-\cos[P_2(t-P_3)]\} & T_0 \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.9}$$

$$V_o(t)=\begin{cases} P_1\{1-\cos[P_2(t-P_3)]\} & T_0 \le t \le T_m \\ P_4\{1-\cos[P_5(t-P_6)]\} & T_m \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.10}$$

Des chercheurs ont proposé différents modèles qui reposent sur la minimisation d'un certain critère. Le profil de vitesse généré par l'équation (1.11) est obtenu en utilisant le critère de Flash et Hogan (1985), soit le "Minimum Jerk". Celui-ci est en fait la

minimisation de la troisième dérivée du déplacement et donne des profils symétriques continus.

$$V_o(t)=\begin{cases} P_1(t-P_2)^2(t-P_3)^2 & T_0\le t\le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.11}$$

Si nous minimisons la quatrième dérivée du déplacement, nous obtenons le critère "Minimum Snap" d'Edelman et Flash (1987). Nous obtenons encore un profil de vitesse symétrique continu, qui est défini par un polynôme de degré sept (1.12).

$$V_o(t)=\begin{cases} P_1(t-P_2)^3(t-P_3)^3 & T_0\le t\le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.12}$$

En utilisant une équation exponentielle pour décrire la trajectoire d'un mouvement, Gutman et Gottlieb (1991) arrivent avec un profil de vitesse de la forme de l'équation (1.13). Ce modèle a cependant une restriction en ce qui concerne l'exposant qui affecte t : il est fixé à 3. Le modèle généralisé, proposé par Gutman et Al. (1992), utilise un paramètre comme exposant, comme le montre l'équation (1.14). Ces deux modèles produisent des profils de vitesse asymétriques et continus.

$$V_o(t)=\begin{cases} P_1(t-P_2)^2 e^{-\frac{(t-P_2)^3}{P_3}} & T_0\le t\le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.13}$$

$$V_o(t)=\begin{cases} P_1(t-P_2)^{(P_3-1)} e^{-\frac{(t-P_2)^{P_3}}{P_4}} & T_0\le t\le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.14}$$

Selon la représentation que propose Plamondon (1989), les trajectoires observées sont en fait le résultat de la production d'un profil de vitesse discontinu ayant une forme de

cloche asymétrique. Il utilise donc deux fonctions gaussiennes pour représenter la vitesse curviligne observée (1.15). Le domaine de la première fonction va du point de départ jusqu'au maximum de vitesse et celui de la seconde, du maximum au point final.

$$V_o(t)=\begin{cases} P_1 e^{-\left(\frac{t-P_2}{P_3}\right)^2} & T_0 \le t \le T_m \\ P_4 e^{-\left(\frac{t-P_5}{P_6}\right)^2} & T_m \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.15}$$

Dans sa thèse, Alimi (1995) a ajouté le modèle n'utilisant qu'une seule fonction gaussienne au lieu d'en utiliser deux. Ce modèle lui a permis d'analyser l'effet de l'asymétrie du profil de vitesse. Le modèle (1.16) donne des courbes symétriques et continues.

$$V_o(t)=\begin{cases} P_1 e^{-\left(\frac{t-P_2}{P_3}\right)^2} & T_0 \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.16}$$

Plamondon (1991) propose une explication sur la génération des profils de vitesse pour des mouvements rapides. Il affirme que la génération d'un mouvement est le résultat d'une activation séquentielle de générateurs de vitesse. Le théorème central limite prédit, dans un tel cas, que la courbe observée sera de forme lognormale. Le modèle lognormal (1.17) produit des profils asymétriques continus.

$$V_o(t)=\begin{cases} \dfrac{P_1}{(t-P_2)} e^{-P_3[\ln(t-P_2)-P_4]^2} & T_0 \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \tag{1.17}$$

Comme nous le savons, une courbe lognormale n'a pas de borne supérieure sur son domaine. Cependant, nous observons qu'un mouvement a un début et une fin. Plamondon

(1992), après plusieurs expérimentations, a observé qu'un modèle lognormal borné (1.18) donnait de meilleurs résultats que celui qui ne l'est pas.

$$V_o(t)=\begin{cases} \dfrac{P_1}{(t-P_2)(P_3-t)} e^{-P_4\left[\ln\left(\frac{t-P_2}{P_3-t}\right)-P_5\right]^2} & T_0 \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \qquad (1.18)$$

Pour fins d'étude, Alimi et Plamondon (1993) ont proposé le modèle lognormal borné non normalisé (1.19). Ils voulaient analyser quel modèle (normalisé ou non normalisé) permettait une meilleure approximation du modèle delta-lognormal.

$$V_o(t)=\begin{cases} \dfrac{P_1(P_3-t)}{t-P_2} e^{-P_4\left[\ln\left(\frac{t-P_2}{P_3-t}\right)-P_5\right]^2} & T_0 \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \qquad (1.19)$$

Après une analyse plus approfondie des mouvements, Plamondon (1993) a proposé que l'activation de deux groupes neuro-musculaires se fait par deux impulsions synchrones. Une impulsion active les muscles agonistes, c'est à dire ceux qui génèrent le mouvement. L'autre active les muscles qui freinent le mouvement, c'est à dire les antagonistes. Cette nouvelle observation, appliquée au modèle lognormal génère le modèle delta-lognormal (1.20). Ce dernier est en fait la différence pondérée entre les deux courbes lognormales générées par les deux impulsions. Celui-ci produit des profils asymétriques continus.

$$V_o(t)=\begin{cases} \dfrac{P_1}{(t-P_2)} e^{-P_3[\ln(t-P_2)-P_4]^2} - \dfrac{P_5}{(t-P_2)} e^{-P_6[\ln(t-P_2)-P_7]^2} & T_0 \le t \le T_1 \\ 0 & \textit{ailleurs} \end{cases} \qquad (1.20)$$

Une version du modèle à deux impulsions utilisant le modèle gaussien a aussi été proposée pour fin d'analyses par Alimi et Plamondon (1993). L'équation (1.21) décrit mathématiquement ce modèle.

$$V_o(t)=\begin{cases} P_1 e^{-\left(\frac{t-P_2}{P_3}\right)^2}-P_4 e^{-\left(\frac{t-P_5}{P_6}\right)^2} & T_0\le t\le T_1 \\ 0 & \textit{ailleurs} \end{cases} \qquad (1.21)$$

Alimi (1995) a aussi proposé d'utiliser d'autres modèles comme l'utilisation de la fonction sigmoïdale continue (1.22), sigmoïdale discontinue (1.23), Beta (1.24), Gamma (1.25) ou Weibull (1.26), mais ceux-ci ne sont pas intéressants dans le cadre de la présente recherche. Nous les présentons donc qu'à titre indicatif. En effet, ces modèles n'ont aucune explication physique et il ne les présente que pour des fins de comparaisons. Ces modèles auraient quand même pu être utilisés pour la modélisation s'il s'était avéré qu'ils avaient obtenu d'excellents résultats lors de la reconstruction des profils de vitesse. Pour être valables il aurait aussi fallu trouver une explication physique à chacun des paramètres ce qui s'avère impossible pour plusieurs modèles.

$$V_o(t)=\begin{cases} P_1\dfrac{t-P_2}{[1+P_3(t-P_2)[P_4]^2} & T_0\le t\le T_1 \\ 0 & \textit{ailleurs} \end{cases} \qquad (1.22)$$

$$V_o(t)=\begin{cases} P_1(t-P_2)^{P_3-1}e^{-(t-P_2)^{P_3}} & T_0\le t\le T_1 \\ 0 & \textit{ailleurs} \end{cases} \qquad (1.23)$$

$$V_o(t)=\begin{cases} P_1\dfrac{t-P_2}{[1+P_3(t-P_2)[P_4]^2} & T_0\le t\le T_m \\ P_5\dfrac{t-P_6}{[1+P_7(t-P_6)[P_8]^2} & T_m\le t\le T_1 \\ 0 & \textit{ailleurs} \end{cases} \qquad (1.24)$$

$$V_o(t)=\begin{cases} P_1(t-P_2)^{P_3}e^{-P_4(t-P_2)} & T_0\le t\le T_1 \\ 0 & ailleurs \end{cases} \quad (1.25)$$

$$V_o(t)=\begin{cases} P_1(t-P_2)^{P_3}(P_4-t)^{P_5} & T_0\le t\le T_1 \\ 0 & ailleurs \end{cases} \quad (1.26)$$

Tous les modèles présentés, jusqu'à présent, permettent de reconstruire le profil de vitesse pour un mouvement simple et rapide. En effet, toutes les équations nous donnent la vitesse en fonction du temps. Ces modèles ne permettent cependant pas directement de reconstruire les mouvements complexes, car ils ne tiennent pas compte de l'aspect vectoriel de la vitesse, mais seulement de son module. Comme tous les modèles sont écrits sous la même forme, il s'agit de mettre la vitesse sous forme vectorielle en ajoutant des paramètres et tous les modèles seront plus ou moins aptes à décrire des mouvements en deux dimensions. En fait, tel que démontré dans la thèse de Alimi (1995), peu de modèles ont une performance semblable à celui utilisant les équations delta-lognormales pour reconstruire les profils de vitesse, car ce dernier est basé sur la physiologie qui permet de générer des mouvements.

Plamondon et Guerfali (1998) ont proposé une extension pour le modèle delta-lognormal. Ils ont ajouté la courbure du mouvement ainsi que l'angle initial du geste pour permettre de connaître la vitesse de manière vectorielle. Ce modèle sera présenté plus en détails dans le deuxième chapitre. Ces deux paramètres pourraient être appliqués à un des modèles précédents pour obtenir un modèle de vitesse vectorielle, mais aucun n'égale les performances du modèle delta-lognormal. Nous nous intéresserons donc plus particulièrement à ce dernier

Finalement, pour le passage à la troisième dimension, nous pouvons utiliser une théorie semblable à celle qui a été utilisée pour le passage en deux dimensions. En effet, si nous trouvons une expression qui permet d'exprimer la vitesse de manière vectorielle en

trois dimensions, un des modèles précédents peut être utilisé. Cependant, seuls les modèles ayant une bonne performance pour la reconstruction sont intéressants pour notre analyse, donc nous étudierons de nouveau le modèle delta-lognormal.

1.3.2 Modèles en deux dimensions divers

Nous allons maintenant présenter d'autres modèles qui sont décrits dans la littérature, mais dont Alimi n'a pas traité dans sa thèse. Un certain nombre de ces modèles reposent sur la minimisation d'un paramètre. Nous avons déjà parlé du minimum jerk et du minimum snap. À ceux-ci nous pouvons ajouter le modèle qui minimise le changement des moments de force de Uno et al. (1989). Ce modèle consiste à minimiser la somme des moments de force au carré pour tout le mouvement. Le critère est présenté dans l'équation (1.27)

$$C_T = \frac{1}{2}\int_0^{t_f} \sum_{i=1}^{n} \left(\frac{dz_i}{dt}\right)^2 dt \tag{1.27}$$

Ce modèle donne des résultats semblables au modèle minimum jerk dans plusieurs conditions expérimentales, mais les résultats diffèrent dans d'autres cas. Les tests comparatifs ont été faits pour des mouvements en deux dimensions. Le modèle minimisant le changement des moments de force permet une meilleure reconstruction des mouvements observés, mais n'explique pas les changements homothétiques.

Okadome et Masaaki (1995) ont utilisé le modèle du minimum jerk comme axiome de base pour leur algorithme de reconstruction des mouvements rapides. Ils disent en fait que toutes les tâches peuvent être spécifiées par le modèle minimum jerk. Si une tâche ne peut être décrite par le modèle, il existe une méthode pour revenir à celui-ci. En segmentant une tâche en sous tâches, ils sont en mesure de reconstruire des mouvements en deux

dimensions. Le problème de ce modèle est qu'il nécessite souvent beaucoup de points du tracé original pour obtenir de bons résultats.

Les modèles basés sur la minimisation d'un paramètre nécessitent des calculs complexes. Pour résoudre le problème Wada et Kawato (1993) proposent d'utiliser un algorithme de réseaux neuronaux pour l'optimisation. Ils utilisent le modèle qui minimise le changement des moments de force pour leurs tests, mais le réseau pourrait être utilisé avec d'autres modèles. L'algorithme de réseaux neuronaux proposé contient un modèle de dynamique direct et inverse. De plus un calcul approximatif du critère de minimisation est nécessaire à chaque itération. Ils ne proposent donc pas un nouveau modèle, mais une nouvelle méthode de trouver les paramètres des modèles existants.

La notion de trajectoire d'équilibre a été proposée par Hogan (1985). Ce modèle dit qu'un mouvement est généré en changeant graduellement le point d'équilibre de la position du membre que l'on désire déplacer. Le point d'équilibre est caractérisé en terme de torques aux articulations. Pour évaluer celui-ci, Flash (1987) a utilisé un modèle masse-ressort pour le bras humain en deux dimensions. Avec quelques manipulations mathématiques, il transforme la trajectoire d'équilibre trouvée en trajectoire réelle. Ce modèle implique une rétroaction continue ce qui n'est habituellement pas le cas pour beaucoup de mouvements rapides.

Une autre forme de trajectoire proposée pour les mouvements est la trajectoire virtuelle. Elle est en fait la représentation mentale du mouvement à faire avant qu'il ne soit exécuté. Ce modèle a été proposé par Katayama et Kawato (1993). La trajectoire virtuelle dépend de la rigidité des articulations. Ils ont donc utilisé le modèle du bras humain à six muscles, illustré à la figure 1.1, pour calculer les trajectoires. Les trajectoires virtuelles sont semblables à celles réelles seulement pour des mouvements lents avec une grande rigidité.

Pour des mouvements rapides, la trajectoire devient complexe et ne peut être utilisée pour prédire le mouvement réel.

Goodman et Gottlieb (1995) proposent d'utiliser des équations différentielles pour reconstruire les mouvements observés. Ils partent d'une équation générale et choisissent les paramètres de manière à ce que les résultats obtenus correspondent aux traces observées. Leur modèle permet de déceler les propriétés qui sont presque invariantes dans l'exécution des mouvements rapides. Ils proposent même l'expression du modèle pour des gestes en trois dimensions.

Le modèle que plusieurs chercheurs ont utilisé est celui du bras humain comprenant six muscles. Le schéma de ce modèle est présenté à la figure 1.1. On voit que tous les muscles sont dans un même plan. Le bras est donc contraint à des mouvements en deux dimensions. Malgré ces restrictions, cette représentation du bras humain a permis de faire plusieurs simulations et de calculer les paramètres de plusieurs modèles. Pour l'analyse des mouvements en trois dimensions, nous devons ajouter des muscles au niveau de l'épaule qui permettent des mouvements perpendiculaires au plan. Cette représentation nécessite cependant beaucoup de paramètres pour caractériser chaque muscle.

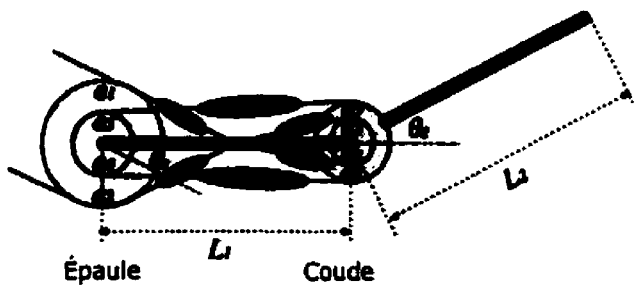


FIGURE 1.1 Modèle du bras humain à six muscles. (Source : Katayama et Kawato, 1993 Fig 1)

Jusqu'à présent, les modèles présentés ont servi principalement à modéliser les mouvements des bras ou des membres supérieurs. Pour leur part, Flashner, Beuter et Arabyan (1988) se sont intéressés aux mouvements des pieds lors de la marche. Ils ont tenté de faire coïncider des fonctions mathématiques périodiques avec les graphiques de positions observées. Ceci fonctionne dans leur cas, car le mouvement est cyclique. Pour notre étude, nous travaillons avec un seul mouvement simple, donc l'utilisation de fonctions périodiques n'est pas utile.

Taga (1995) a aussi travaillé sur le mouvement de la marche. Il a intégré les équations du mouvement dans un réseau neuronal. Ce réseau reçoit des impulsions et génère une suite de position des jambes. Ce qui active celui-ci est un neurone qui génère un rythme cyclique. Ce réseau ne nous donne donc pas un modèle général pour les mouvements, mais sert seulement à reproduire les mouvements de marche. Nous voyons qu'il y a utilisation d'un générateur d'impulsion cyclique, ce qui n'est pas nécessaire pour modéliser des mouvements rapides.

Le concept de réseau cinématique (K-net) a été proposé par Mussa Ivaldi, Morasso et Zaccaria (1988). Ce type de réseau ne permet pas de reconstruire directement des mouvements, mais donne plutôt des classes de mouvements comme solution. Ils permettent de rendre explicite les relations entre les éléments des structures complexes. Un K-net est un graphe directionnel où les liens représentent des relations causales et les noeuds, des forces ou des déplacements généralisés. Comme le but de cette méthode n'est pas de générer une solution unique, il n'est pas possible de l'utiliser pour la modélisation directement.

1.3.3 Modèles en trois dimensions

Tous les modèles qui précèdent concernent les mouvements en deux dimensions. Cependant, comme nous voulons faire de la modélisation en trois dimensions, il faut voir comment chacun des modèles précédents pourraient être étendus à la troisième dimension. Ces extensions seront présentées dans le troisième chapitre de ce mémoire.

Dans la littérature nous avons quand même trouvé deux modèles de mouvements en trois dimensions. Le premier a été proposé par Costa et al. (1997) et il consiste en l'utilisation de réseaux de neurones pour reproduire les mouvements humains. Le réseau est tout d'abord entraîné avec des centaines de mouvements. Par la suite, il donne la position 3D du membre à raison d'un point par itération. Il reproduit des mouvements qui sont semblables à ceux des humains.

Morasso (1983) propose une seconde méthode de décrire les mouvements. Il utilise les équations de Frenet et la géométrie analytique. Il calcule donc le module du vecteur vitesse, la courbure et la torsion des mouvements observés. Il modélise un mouvement complexe par la somme de mouvements simples, c'est à dire des mouvements ayant un profil de vitesse en forme de cloche asymétrique. Il ne donne pas d'équation pour les profils observés, mais il réussit à reconstruire le mouvement en intégrant les équations de Frenet. Son but n'était pas de faire de la modélisation, mais de voir comment se comportent les profils de vitesse, courbure et torsion. Il en vient à la conclusion que les pics de courbure correspondent aux minimums de vitesse. Il n'a pas trouvé de règle pour les pics de torsion, car ils arrivent à n'importe quel moment. Nous reviendrons sur cette observation au troisième chapitre.

Dans cette section, nous avons présenté les principaux modèles que nous avons rencontrés dans la littérature. Comme nous pouvons le constater, le nombre de modèles

pour les mouvements en trois dimensions est très restreint. Il reste donc beaucoup de travail à faire dans ce domaine. La section suivante décrira le travail réalisé dans le cadre de cette recherche ainsi que le contenu des différents chapitres faisant partie du présent mémoire.

1.4 Plan du mémoire

1.4.1 Travail qui a été réalisé

Dans ce mémoire, nous allons nous concentrer surtout sur l'élaboration d'un modèle général pour les mouvements rapides en trois dimensions. Nous programmerons un extracteur de paramètres pour le modèle. Nous allons nous concentrer à concevoir un extracteur qui nous permettra de trouver les valeurs des paramètres recherchés. Par contre, nous allons peu nous soucier du temps nécessaire pour l'extraction. En effet, nous voulons prouver que notre modèle est fonctionnel et pour cela l'extraction en temps réel n'est pas nécessaire.

Nous allons aussi commencer à construire une banque de gestes. Il est certain que cette banque de données ne sera pas énorme, mais elle nous permettra de faire des tests sur des gestes simples. Elle proviendra d'un petit groupe de personnes travaillant à l'École Polytechnique de Montréal. Cette banque pourrait servir de base à l'élaboration d'une banque de données de référence qui pourrait être disponible via l'internet ou sur cédérom.

Nous allons enfin travailler sur la comparaison des gestes. À partir de notre banque de gestes simples, nous allons tenter de voir sommairement s'il est possible de différencier certains gestes. Comme les tests ont été faits sur un nombre restreint de personnes, il faudra faire attention à l'interprétation des résultats. Nous allons donc nous contenter de faire des observations préliminaires et comparer les profils de vitesse observés.

Nous avons soumis deux articles pour publication dans des conférences. Le premier a été présenté à "Progress in motor control III" (Leduc et Plamondon, 2001a) et le second à "International Graphonomics Society" (Leduc et Plamondon 2001b). Les deux articles traitent du modèle delta-lognormal en trois dimensions tel que décrit dans ce mémoire.

Nous ne traiterons pas des applications du modèle, car celles-ci dépassent largement le cadre du projet. Nous allons plutôt nous concentrer sur les aspects théoriques de celui-ci. Le présent mémoire comporte donc cinq chapitres incluant celui-ci. Le travail réalisé sera donc présenté selon le plan ci-dessous.

1.4.2 Contenu des chapitres subséquents

Dans le second chapitre, nous ferons une revue du modèle delta-lognormal en une et deux dimensions. Nous présenterons les bases du modèle ainsi que les résultats qui ont été obtenus avec celui-ci. Nous ferons une brève revue du fonctionnement de l'extracteur de paramètres en gardant toujours à l'esprit la généralisation de celui-ci à la troisième dimension. Les résultats obtenus sur des signatures seront montrés.

Le nouveau modèle sera présenté au chapitre trois. Avant de présenter ce modèle, nous allons proposer des extensions possibles pour les modèles existants et nous verrons les raisons pour lesquels ils n'ont pas été retenus. Le modèle delta-lognormal vectoriel généralisé sera ensuite présenté. Des simulations, démontrant la reconstruction possible de gestes simples, seront montrées. Comme les paramètres devront être éventuellement extraits d'une manière automatique, quelques principes généraux qui devront être pris en compte lors de l'élaboration de celui-ci, sont mentionnés.

Dans le quatrième chapitre nous présenterons les différents résultats obtenus et nous validerons le modèle. Nous allons tout d'abord décrire la composition de la base de

données utilisée pour nos tests. L'équipement ainsi que les logiciels utilisés seront expliqués. Nous ferons une brève description du programme qui a été écrit pour nous permettre d'avoir une représentation visuelle des résultats. Nous présenterons ensuite les résultats obtenus pour des mouvements simples. Une comparaison basée sur les courbes de vitesse suivra dans le but d'explorer s'il est possible de croire que notre modèle pourrait permettre de différencier des personnes ou des gestes. Le chapitre contiendra enfin une discussion critique sur le modèle présenté et le protocole expérimental utilisé.

Le dernier chapitre servira de conclusion au travail. Il contiendra d'abord une synthèse du travail réalisé. Une analyse critique fera ressortir les points forts et les points faibles du modèle ainsi que ceux concernant le projet en général. De plus de nouvelles pistes de recherche seront proposées ainsi que des applications probables du modèle delta-lognormal généralisé.

Chapitre 2

Théorie cinématique en deux dimensions

2.1 Introduction

Dans le chapitre précédent, nous avons présenté un grand nombre de modèles permettant de reconstruire les mouvements humains. Le présent chapitre se consacre exclusivement à l'un de ces modèles soit le modèle delta-lognormal vectoriel (Plamondon et Guerfali, 1998). Nous passerons en revue toutes les connaissances que nous avons à propos de celui-ci. Nous verrons premièrement les origines du modèle en une dimension. Nous expliquerons comment nous sommes arrivés au modèle en deux dimensions. Ce dernier nous permet de trouver des paramètres qui caractérisent l'écriture manuscrite. Un extracteur automatique de paramètres sera aussi présenté. Tout au long du chapitre, des exemples de résultats, qui ont été obtenus, seront montrés. Nous débutons donc par le modèle delta-lognormal en une dimension.

2.2 Origines du modèle delta-lognormal

Au cours du dernier siècle, des chercheurs se sont intéressés à la génération des mouvements rapides. Ils se sont aperçus que pour ce type de geste, nous observons un profil de vitesse sous forme de cloche asymétrique. Les différents profils se superposent assez bien, et même presque parfaitement dans certains cas, selon les conditions expérimentales, après une mise à l'échelle du temps et de l'amplitude. Des chercheurs ont donc formulé plusieurs modèles pour tenter de reproduire ces observations. Cette section présentera les origines du modèle delta log-normal en une dimension (Plamondon, 1995a,b).

Nous verrons les bases physiologiques et mathématiques de celui-ci. L'extension en deux dimensions sera expliquée dans la section suivante et sera suivie par la présentation de l'extracteur automatique de paramètres.

2.2.1 Implications physiologiques

Plamondon (1995a,b) s'est intéressé au système physiologique de génération des mouvements et s'est basé sur ses observations pour proposer son modèle. Nous reprenons ici l'explication de la méthode de génération d'un geste telle que présentée par Plamondon (1995a). On sait que pour activer un muscle il faut une cascade d'activations de systèmes neuro-musculaires. Ces systèmes interagissent aussi bien de manière sérielle que de manière parallèle. Au sommet de la hiérarchie de contrôle, nous trouvons le cortex moteur primaire, le cortex pré-moteur et la zone motrice supplémentaire. Ces zones reçoivent des informations de la périphérie du corps par l'intermédiaire des capteurs sensoriels. Elles contiennent aussi les cartes somatotopiques.

Le second niveau de la hiérarchie est constitué du tronc cérébral. Celui-ci comporte trois systèmes neuronaux qui sont le médial, le latéral et l'aminergique. Il sert à moduler l'activation des neurones moteurs et les interneurones qui font partie de la moelle épinière. Cette dernière fait partie du niveau suivant dans la hiérarchie. Les muscles à proximité ou distants sont activés directement ou indirectement par ces neurones moteurs. Il ne faut pas non plus oublier le cervelet et le ganglion basal. Ceux-ci servent de régulateurs aux fonctions motrices. Dans les niveaux les plus bas de la hiérarchie, on retrouve les réseaux musculo-squelettiques.

Lorsqu'un système d'une telle complexité travaille de manière ordonnée nous disons qu'il y a synergie. Nous savons que lors de l'exécution d'un geste rapide deux systèmes neuro-musculaires s'activent. Nous avons en fait une activation du système agoniste, qui

permet d'exécuter le mouvement voulu. Si seul le système agoniste existait, les gestes auraient théoriquement une amplitude infinie, ce qui est impossible en pratique, car il nous faudrait des muscles d'une longueur infinie. Nous avons donc un système antagoniste qui permet de freiner le mouvement. Ces deux systèmes s'activent de manière synchrone, mais ont des temps de réponse différents. De plus, pour un mouvement rapide, on peut considérer que la rétroaction visuelle est inexistante, ce qui conduit à un système en boucle ouverte. Cette synergie peut être représentée comme à la figure 2.1.

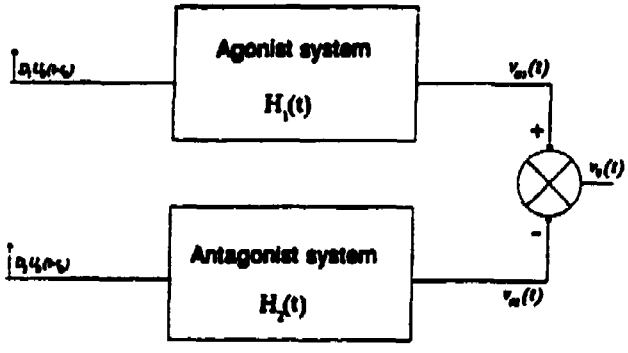


FIGURE 2.1 Modèle de la synergie. Représentation des systèmes neuro-musculaires agoniste et antagoniste formant la synergie (Source : Plamondon 1995a Fig 1.A)

Dans cette figure nous voyons bien les deux systèmes agonistes et antagonistes qui sont chacun activés par une impulsion. Ces impulsions arrivent simultanément au temps t_0, mais ont des amplitudes différentes, soit D_1 et D_2. Le module de vitesse observé est la différence pondérée entre la réponse impulsionnelle des deux systèmes. Un tel système peut être décrit par l'équation (2.1) suivante :

$$v_o(t)=D_1H_1(t-t_0)-D_2H_2(t-t_0) \qquad (2.1)$$

2.2.2 Élaboration du modèle de base

Nous pouvons donc calculer le module de vitesse si nous trouvons une expression pour $H_i(t-t_0)$. Une première hypothèse faite est que tous les sous-systèmes sont connectés de manière sérielle. La figure 2.2 nous montre schématiquement ces connections. Nous voyons que chaque sous-système est connecté au suivant. Si nous considérons que chacun de ceux-ci a une réponse indépendante et que leur nombre est suffisamment élevé, nous pouvons appliquer le théorème de la limite centrale. Dans un cas comme celui-ci le théorème dit que la réponse observée devrait être de forme gaussienne.

Plusieurs expériences ont été faites avec un modèle gaussien et les résultats étaient acceptables (Leclerc, 1989; Alimi, 1995). Par contre, comme la courbe gaussienne est symétrique et le profil observé est asymétrique, il fallait utiliser deux courbes gaussiennes pour obtenir des résultats acceptables. Une des approches analysées consistait en l'utilisation d'une gaussienne pour modéliser la montée et d'une autre pour modéliser la descente du profil de vitesse. Cette approche s'est avérée plus performante par rapport à l'utilisation d'une gaussienne simple, car elle permet de tenir compte de l'asymétrie, mais elle produit des profils discontinus.

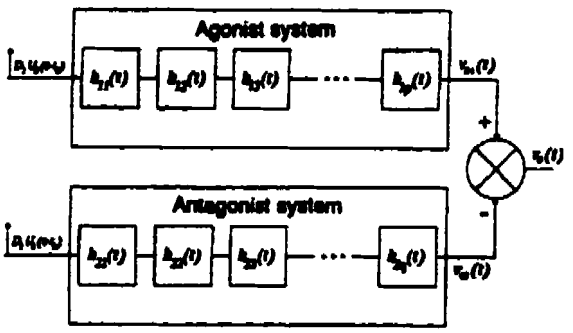


FIGURE 2.2 Synergie faite de deux systèmes séquentiels. Les systèmes agoniste et antagoniste sont formés de sous-systèmes qui interagissent de manière séquentielle. (Source: Plamondon 1995a Fig 1.B)

À cause des problèmes reliés au modèle gaussien, le modèle a été revu et corrigé. Nous savons que les interactions entre les sous-systèmes ne se font pas seulement de manière sérielle, mais il y a aussi de la propagation en parallèle et il peut y avoir du feed-back local. Ce nouveau système est représenté par la figure 2.3. On voit que les systèmes sont reliés à tous les suivants et non pas seulement à leur voisin.

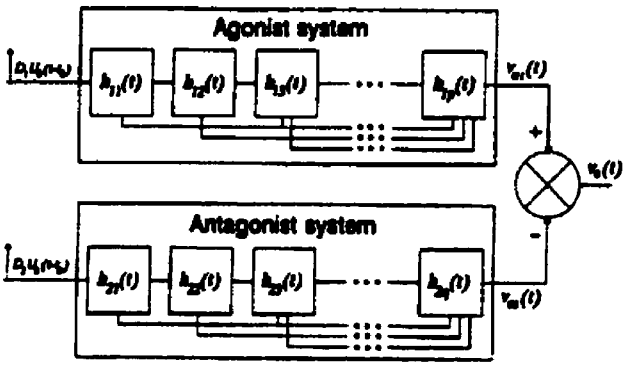


FIGURE 2.3 Synergie faite de deux systèmes à couplage complexe. Chaque système contient des sous-systèmes qui sont couplés de manière complexe et hiérarchique. (Source: Plamondon 1995a Fig 1.C)

En supposant un effet proportionnel entre les temps de délais, le théorème central limite prédit alors, pour une telle configuration, que le logarithme de la réponse observée sera de type gaussien. En faisant un changement de variable, nous trouvons une réponse impulsionnelle de type lognormale. Cette réponse s'applique à chacun des systèmes neuro-musculaires, ce qui nous donne un profil de vitesse de forme delta-lognormale. Ce système est représenté par l'équation (2.2) :

$$v_\sigma(t) = D_1\Lambda(t;t_0,\mu_1,\sigma_1^2) - D_2\Lambda(t;t_0,\mu_2,\sigma_2^2)$$
$$\Lambda(t;t_0,\mu,\sigma^2) = \frac{e^{\frac{-(\ln(t-t_0)-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}(t-t_0)} \qquad (2.2)$$

Le paramètre σ représente le temps de réponse du système sur une échelle logarithmique. On peut le voir comme reflétant la durée pendant laquelle la fonction delta-lognormale est significativement différente de 0. Le paramètre μ représente, pour sa part, le délai de réaction du système sur une échelle logarithmique ce qui reflète le temps qu'il faut pour atteindre le maximum de vitesse. En ajustant les sept paramètres du modèle on réussit à reconstruire les différents profils de vitesse observés. La figure 2.4 montre des exemples de courbes obtenues

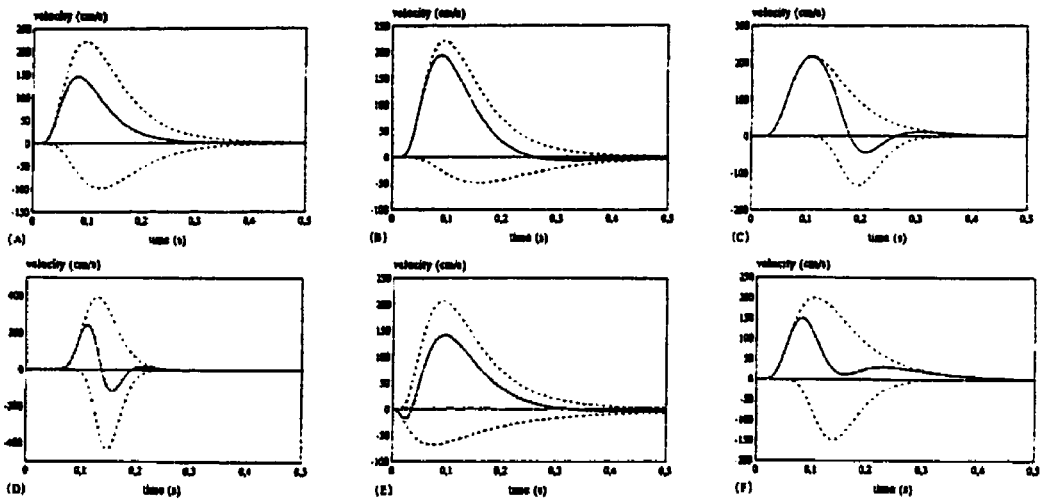


FIGURE 2.4 Profils de vitesse delta-lognormal typiques. Les réponses des deux systèmes sont en pointillé et la courbe delta-lognormal associée est en traits pleins. Le modèle peut former des courbes à un (A), deux (B) ou trois (C) pics de vitesse. Il peut aussi produire des profils ayant une inversion de l'asymétrie (D), avec un petit pic au départ (E) ou des profils oscillants strictement positifs (F). (Source: Plamondon 1995a Fig 2)

Sur cette figure nous voyons en pointillé les deux courbes lognormales qui ont servi à générer les profils delta-lognormaux en traits pleins. Ce modèle permet de générer des profils ayant un, deux ou trois pics de vitesse. Il permet d'expliquer les pics apparaissant lors des mouvements rapides simples. La figure 2.5 montre d'excellents résultats obtenus pour la reconstruction des profils de vitesse observés.

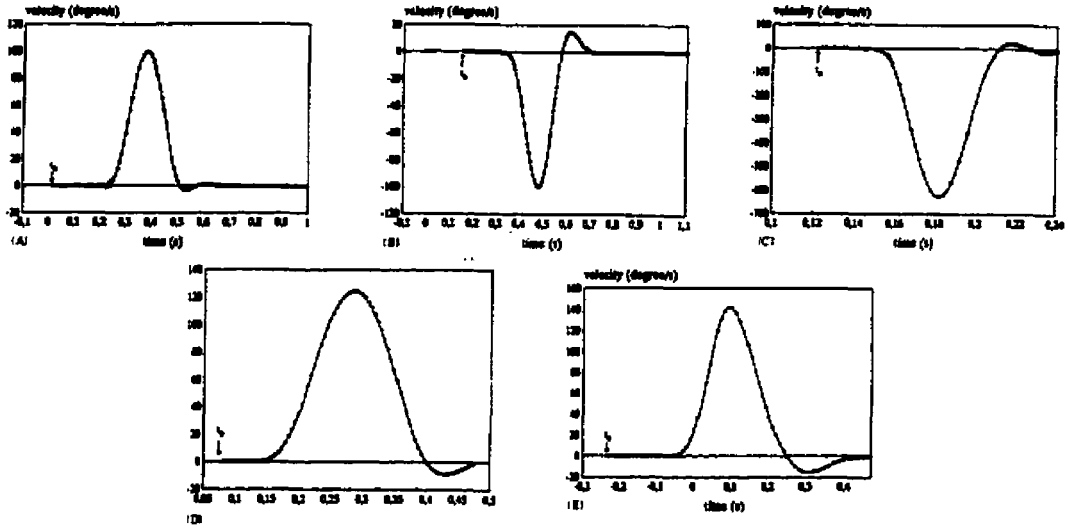


FIGURE 2.5 Résultats obtenus en utilisant le modèle delta-lognormal. Cinq exemples de reconstruction de courbes de vitesse par le modèle delta-lognormal. A. Flexion du poignet d'un singe. B. Extension d'un poignet de singe. C. Saccade visuelle humaine. D. Mouvement d'un bras humain. E. Mouvement d'une tête humaine. (Source : Plamondon 1995a Fig 7)

2.3 Modèle delta-lognormal en deux dimensions

La section précédente a brièvement présenté le modèle delta-lognormal en une dimension et ses origines biologiques. Les résultats prometteurs, obtenus lors de l'analyse de mouvements en une dimension, ont amené Plamondon et Guerfali (1998) à étendre le modèle à la deuxième dimension pour être en mesure de travailler avec des signatures et de l'écriture. En effet, jusqu'à présent nous n'étions capables que de traiter des profils n'ayant pas de superposition de mouvements, par exemple des traits simples rectilignes. Malheureusement, il est rare de voir écrire des personnes sans qu'elles ne combinent des commandes musculaires pour générer des tracés complexes. Ceci se traduit par une superposition de courbes delta-lognormales. Cette section présente donc les fondements du modèle en deux dimensions qui a été utilisé pour extraire des paramètres sur des

signatures et de l'écriture. La section suivante présentera le logiciel utilisé pour extraire ces paramètres et sera suivie par les résultats obtenus à l'aide de cet extracteur.

Dans la figure 2.6 on voit un exemple typique de courbe de vitesse pour le mot "elle". On remarque que l'on a de la superposition, car un mouvement simple démarre sans attendre que le précédent ne soit terminé. Le problème est que cette superposition est vectorielle. En effet, les déplacements se font dans le plan dans des directions différentes. Comme c'est la vitesse tangentielle qui nous intéresse, celle-ci dépend de la vitesse vectorielle de tous les mouvements qui l'ont engendrée. Nous ne pouvons donc pas seulement faire une addition scalaire de deux profils de vitesse point à point directement.

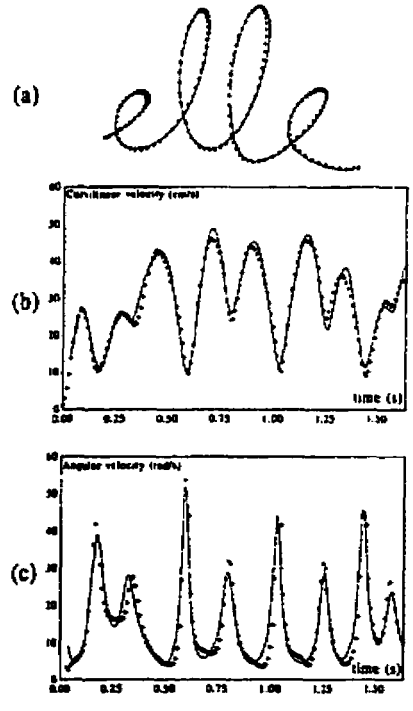


FIGURE 2.6 Exemple de reconstruction en 2D. Les lignes continues sont les données recueillies. Les carrés sont la reconstruction par le modèle delta-lognormal: a) Le mot "elle"; b) Vitesse curviligne correspondante; c) Vitesse angulaire. (Source : Plamondon et Guerfali 1998 Fig 2)

Pour résoudre ce problème, Plamondon et Guerfali (1998) ont fait appel au concept de cibles virtuelles. Avant d'expliquer ce concept, il faut définir le trait simple. Celui-ci se défini comme étant un mouvement ayant une courbure constante et un profil de vitesse de forme delta-lognormale. On voit un exemple de ce type de mouvement dans la figure 2.7. Le tracé a une courbure presque constante et le profil de vitesse ne comporte qu'un maximum de vitesse.

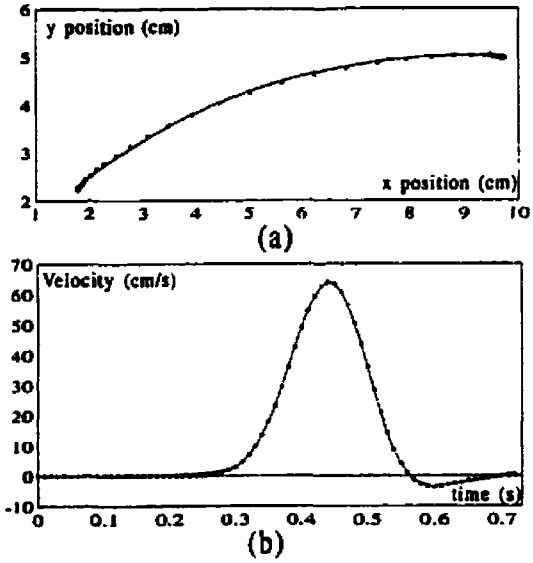


FIGURE 2.7 Exemple de mouvement simple. Traits pleins : mouvement original, carrés: prédiction du modèle. a. Mouvement ayant une courbure constante. b. Profil de vitesse de forme delta-lognormal. (Source : Plamondon et Guerfali 1998 Fig 1b,c)

Pour chaque geste simple, nous devons ajouter deux paramètres aux sept que nous avions déjà. En effet, ce type de geste a une courbure caractéristique, C_0, et une direction initiale, θ_0. Avec ce groupe de neuf paramètres pour chaque geste simple, on est en mesure de reconstruire des mouvements complexes en deux dimensions. L'équation 2.3 nous montre la relation entre l'angle à un temps donné et la courbure du geste.

$$\theta(t)=\theta_0+C_0\int_{t_0}^{t}|\vec{v}(\tau)|d\tau \qquad (2.3)$$

Ayant défini un geste simple, nous pouvons décrire l'écriture comme étant une superposition vectorielle de ceux-ci. La cible virtuelle est définie comme étant le point final du geste simple, s'il n'y a pas eu de superposition. Pour bien comprendre comment se produit l'addition vectorielle, supposons qu'une paire de commandes, D_{i1} et D_{i2}, est transmise aux systèmes neuro-musculaires pour produire un premier mouvement simple. La longueur du trait produit est prédite par D_{i1} - D_{i2} alors que sa durée est estimée par D_{i1}/D_{i2}. Le sujet peut donc amorcer un second trait, partant de la cible virtuelle à atteindre, comme s'il y était déjà. Ceci permet de produire des tracés ayant des formes qui varieront selon le degré de superposition.

La figure 2.8 nous montre le mot "she" ainsi que son profil de vitesse. On voit aussi, à droite du mot original, une suite de traits simples qui ont servi à écrire le mot. Ces traits sont ceux que l'on aurait observés s'il n'y avait eu aucune superposition. Les points numérotés sont donc les cibles virtuelles qui ont permis de construire le mot. Nous voyons enfin un graphique montrant le début et la fin de chaque mouvement simple tel que nous l'avons estimé. Nous nous apercevons donc qu'à certains moments plusieurs gestes simples se superposent.

Plamondon et Guerfali (1998) ont fait plusieurs simulations avec seulement deux traits simples pour montrer le genre de profils que l'on peut observer. Ils ont choisi un moment différent pour l'activation du deuxième geste. La figure 2.9 nous montre des exemples. On voit que plus l'activation du deuxième geste simple arrive tard, plus on est en mesure de distinguer facilement les deux courbes de vitesse. De plus, on remarque que le pic de courbure diminue à mesure que la superposition arrive tôt. Ceci est compréhensible, car plus le second geste se déclenche tôt, plus les courbes de vitesse se

superposent jusqu'au point où elles se confondent complètement et on a l'impression d'avoir affaire à un geste simple.

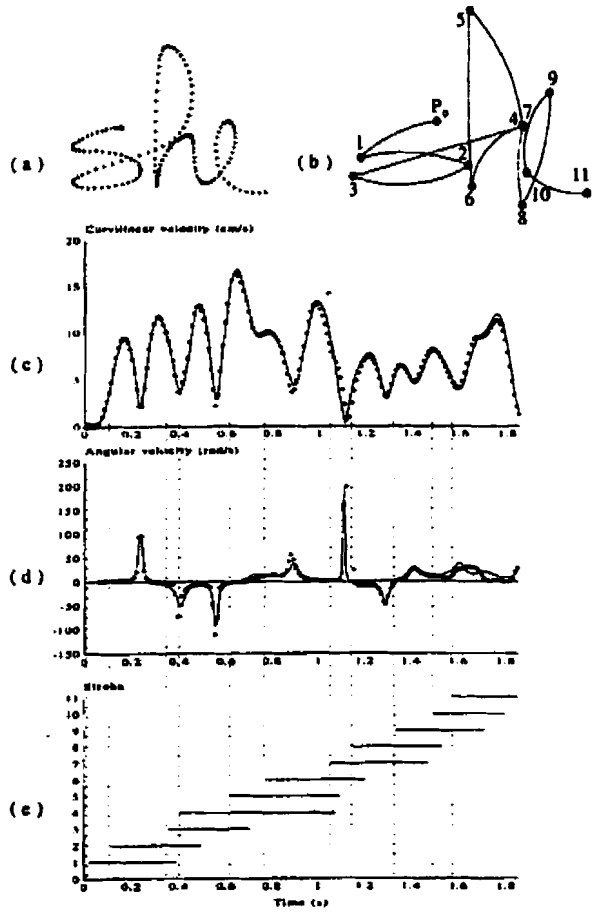


FIGURE 2.8 Reconstruction du mot "she". Traits pleins : trace originale, carrés: reconstruction à l'aide du modèle. a. Mot "she". b. Mouvements simples et cibles virtuelles probablement utilisés lors de l'écriture. c. Profil de vitesse associé. d. Vitesse angulaire. e. Position temporelle des traits simples. (Source : Plamondon et Guerfali 1998 Fig 3)

À l'aide de cette théorie Plamondon et Guerfali (1998) ainsi que Leclerc (1996) ont réussi à reconstruire des mots et des signatures de manière assez fidèle. Le modèle delta-lognormal vectoriel semble surpasser tous les autres, jusqu'à présent, en ce qui trait à la

reconstruction de l'écriture manuscrite et des signatures. Mais, jusqu'ici nous n'avons parlé que de la théorie. En effet, tous les tests et toutes les simulations présentés dans ce chapitre ont été faits manuellement. Ceci veut donc dire qu'il faut que l'extraction de paramètres soit faite par un utilisateur expérimenté qui connaît l'influence que chacun des paramètres a sur la forme du tracé reconstruit. Comme ceci n'est pas souhaitable si nous désirons utiliser le modèle dans différentes applications, il nous a fallu travailler sur un extracteur automatique de paramètres.

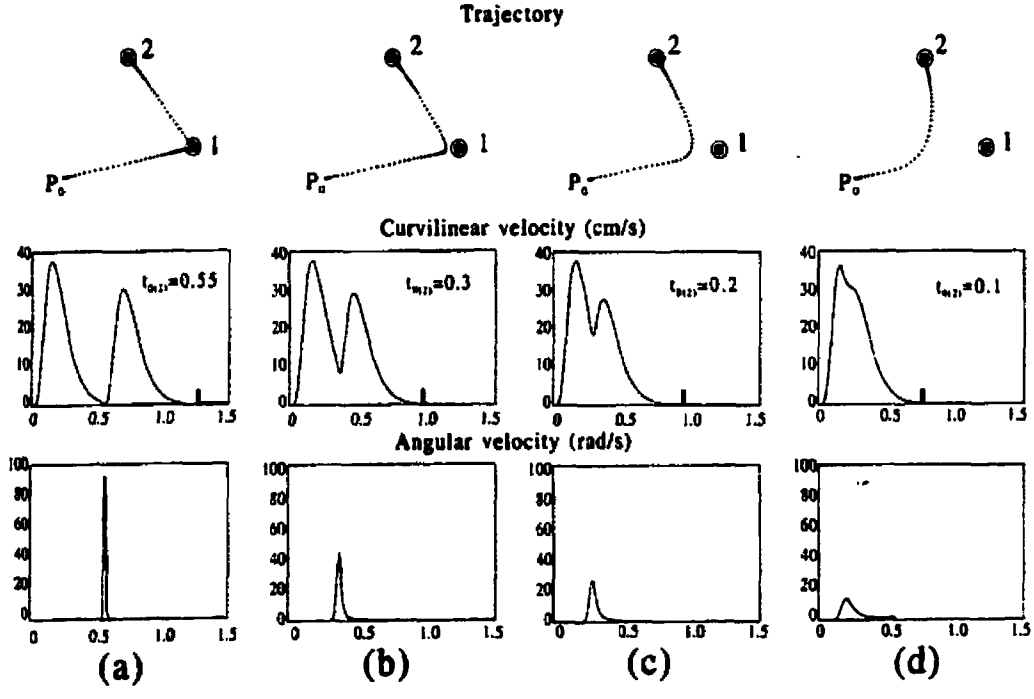


FIGURE 2.9 Superposition de mouvements simples. a. Deux mouvements simples sans superposition. b. Effet d'une légère superposition temporelle. c. Effet d'une superposition modérée. d. Superposition presque totale. (Source : Plamondon et Guerfali 1998 Fig 4)

2.4 Extracteur de paramètres

Dans la section précédente, nous avons aussi expliqué les origines du modèle en deux dimensions et montré quelques applications reliées à l'étude de l'écriture manuscrite. La présente section présente l'extracteur automatique qui a été programmé dans le but de trouver les paramètres du modèle pour des traces aussi complexes que des signatures. Nous verrons en détails les traitements que celui-ci doit exécuter afin d'arriver à un résultat raisonnable.

2.4.1 Traitements préliminaires

L'extracteur de paramètres a été élaboré dans le cadre d'un projet de fin d'études par Leduc (1998). Celui-ci explique en détails toutes les méthodes utilisées ainsi que les résultats obtenus. Ici nous ferons un résumé du fonctionnement de l'extracteur en gardant à l'esprit que celui-ci pourra être utilisé pour l'extraction de paramètres sur des gestes en trois dimensions.

Le premier traitement que l'extracteur doit faire sur une signature est de la segmenter en composantes. En effet, pour une signature, à l'opposé d'un geste à trois dimensions, il arrive que le crayon ne touche pas à la tablette et que les points lus soient inexacts. Ceci arrive dans la plupart des signatures, car il est rare qu'une personne signe sans lever le crayon. Il a donc fallu analyser le fichier de données contenant la signature et éliminer les points qui ont été pris lorsqu'il n'y avait pas de contacts. Nous obtenons plusieurs composantes qu'il faut traiter indépendamment. Pour chacune, nous obtiendrons un certain nombre de groupes de neuf paramètres correspondant au nombre de traits simples dans ce groupe.

Une fois que notre signal a été segmenté, il faut connaître le nombre de traits simples présents pour être en mesure de faire les traitements suivants. Pour ce faire, nous faisons le décompte du nombre de pics de maximum de vitesse dans le profil observé. Avant d'être traité, le signal de vitesse est filtré à 16 Hz, à l'aide d'un filtre de Chebychev de second ordre ayant une atténuation de 40 dB, pour éliminer le bruit et permettre de garder un maximum d'informations sur le signal de vitesse, qui, pour des mouvements humains, a une fréquences maximale inférieure à 16 Hz. Pour déterminer la position et le nombre de pics nous utilisons les dérivées premières et secondes du signal. Cette méthode nous permet de détecter les mouvements qui ne sont pas trop superposés.

Il arrive que la superposition soit trop grande et que seuls des points d'inflexions soient détectés. Pour détecter des cas semblables, nous faisons une analyse de ces derniers entre deux extremums consécutifs. S'il y a plus d'un point d'inflexion, nous déterminons si ce phénomène est provoqué par des gestes qui sont superposés à un point tel qu'il nous est impossible de détecter des maximums distincts. Une fois que nous avons trouvés tous les gestes simples probables, nous nettoyons le début et la fin du signal du bruit résiduel présent. Pour ce faire, nous enlevons les petits maximums de vitesse qui sont dûs aux petits tremblements de la main avant et après le geste voulu, lorsque la main devrait normalement rester immobile.

2.4.2 Estimation de la courbure et de l'angle initial

Nous connaissons le nombre et la position des pics de vitesse ou le nombre de gestes simples. Il nous est donc possible de commencer à estimer des paramètres. Le premier paramètre que nous pouvons calculer est la courbure du mouvement. Pour ce faire nous avons calculé le centre du cercle qui passe par le point du maximum de vitesse et qui épouse le mieux les points adjacents avec la méthode des moindres carrées. En effet, nous

considérons que la courbure est moins affectée par la superposition au point de maximum de vitesse. L'inverse du rayon trouvé nous donne la courbure estimée.

Pour trouver l'angle initial du mouvement, il faut trouver l'angle tangentiel. Ce dernier est estimé en radians à partir du calcul de dy/dx. Ces estimations initiales ne peuvent pas être utilisées pour l'optimisation, car elles sont encore loin de la solution optimale. Pour régler le problème, nous avons fait une recherche itérative dans une matrice vingt par vingt où les lignes représentent différentes valeurs d'angles et les colonnes, les valeurs de courbure. Le couple donnant une erreur minimale quant à la position de l'arc de cercle a été gardé et utilisé dans une seconde itération avec une matrice ayant un pas plus fin. Notre calcul d'erreur tient compte de la position des points et des points d'inflexion qui sont contenus dans le tracé en deux dimensions. Il est certain que cette erreur devra être évaluée autrement en trois dimensions. Ceci nous a permis de trouver des valeurs acceptables pour les paramètres et celles-ci seront utilisées pour l'étape d'optimisation. L'écart entre les paramètres estimés et ceux optimaux dépend du geste analysé et il nous a été impossible de l'évaluer.

2.4.3 Estimation de D_1, μ_1 et σ_1

Avec les courbures trouvées, nous avons les cercles tangents aux maximums de vitesse. Ces cercles décrivent les mouvements simples et la rencontre de deux cercles qui appartiennent à deux maximums de vitesse consécutifs permet de définir la position d'une cible virtuelle. Lorsque nous connaissons la position de ces cibles, nous trouvons facilement le paramètre D et θ_0. En effet, il suffit d'utiliser la géométrie du cercle. Cependant, il arrive que les deux cercles ne se rencontrent pas à cause d'erreurs dans l'estimation des courbures. Dans un tel cas nous prenons le point milieu entre les deux cercles. Dans le cas où il y aurait deux points de rencontre, il faut trouver le point le plus probable.

Quelle que soit la cible choisie, il faut qu'elle soit validée. Nous vérifions entre autres que la cible est à l'extérieur du tracé. De plus il faut que la cible soit à une distance acceptable du tracé. Il est fort improbable qu'à la suite de la superposition de deux mouvements la cible se retrouve à plusieurs dizaines de centimètre de la trace. On tient cependant compte du degré de superposition des traits. Le dernier test permet de voir si l'arc de cercle ne couvre pas une trop grande plage angulaire. En effet, il est peu probable qu'un humain fasse un geste de plus de quatre-vingt-dix degrés d'amplitude étant donné le type de mouvements utilisés pour générer de l'écriture et des contraintes imposées par la taille de la tablette. Si la position de la cible donne un résultat négatif à un de ces tests, la cible est positionnée au minimum de vitesse. Nous réajustons ensuite les paramètres pour tenir compte des nouvelles positions. Tous ces calculs devront être modifiés pour l'extracteur en trois dimensions.

À partir de l'estimation de ces trois paramètres, il nous est possible d'en estimer trois autres qui caractérisent le profil de vitesse. Pour estimer ces derniers par une méthode graphique (Guerfali et Plamondon, 1993), il faut que chaque courbe delta-lognormale soit isolée. Comme ce n'est pas le cas à cause de la superposition, nous remédions à ce problème, en enlevant au signal original tout ce qui est connu pour ne laisser que ce qui est inconnu et la courbe à optimiser. La figure 2.10 montre un exemple. En effet, en soustrayant L1 et L3 nous isolons L2. Nous traitons aussi les courbes dans un certain ordre pour réduire au maximum l'effet de superposition.

L'estimation des paramètres se fait par la méthode graphique présentée par Guerfali et Plamondon (1993). Cette méthode utilise les pentes de la courbe aux points d'inflexions ainsi que le triangle formé pour estimer les paramètres μ, σ, t_0 et D. Elle fonctionne bien dans la plupart des cas. Il faut cependant faire attention lorsque la courbe est presque symétrique, c'est-à-dire que le ratio des pentes est près de 1, car les valeurs obtenues peuvent être erronées, par exemple, t_0 devient très négatif.

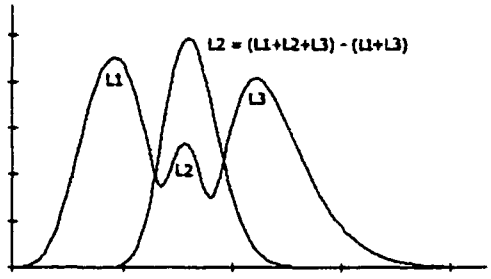


FIGURE 2.10 Soustraction de courbes connues. L1, L2 et L3 sont le résultat d'une addition vectorielle. En soustrayant de façon vectorielle L1 et L3 du profil de vitesse, il nous reste L2 que nous voulions estimer.

Dans notre cas, nous avons gardé la valeur de D que nous avions estimé à l'aide des cibles virtuelles, car nous considérons qu'elle est une meilleure estimation que celle obtenue par la méthode graphique. Pour les autres paramètres, nous utilisons ceux de l'estimation graphique à moins que la valeur du paramètre t_0 soit inacceptable. Nous le déclarons inacceptable s'il se trouve trop loin du maximum de vitesse auquel il est associé. Dans ce cas, nous fixons t_0 à une certaine distance du maximum de vitesse qui dépend de la superposition et nous ajustons μ à -1,5 et σ à 0,25, soit des valeurs moyennes observées. Après chaque estimation nous faisons une première optimisation grossière des paramètres. Ceci a pour but d'avoir une solution initiale acceptable avant la véritable phase d'optimisation.

2.4.4 Estimation de D_2, μ_2 et σ_2

Comme nous voulons extraire des paramètres pour des courbes delta-lognormale, soit la différence entre deux courbes lognormales, et que nous avons une estimation des paramètres pour une courbe lognormale, il faut faire un traitement supplémentaire avant l'optimisation finale. Pour passer des paramètres d'une lognormale à ceux d'une delta-lognormale, nous avons utilisé quelques observations. Par exemple, nous savons que le

mouvement antagoniste est de petite amplitude, donc nous avons choisi une amplitude dix fois plus petite que la courbe lognormale initiale. Nous savons aussi que les courbes agoniste et antagoniste auront sensiblement la même forme mais le délai du système antagoniste sera plus long. Nous trouvons donc une solution qui peut être considérée acceptable comme point de départ.

2.4.5 Optimisation de tous les paramètres

L'optimisation se déroule en cinq phases. Il s'agit en fait d'exécuter cinq fois une boucle contenant deux traitements, soit l'optimisation des paramètres de la courbe de vitesse et l'optimisation de la courbure et de l'angle initial. L'optimisation pour le premier traitement se fait avec la méthode d'optimisation non linéaire Levenberg-Marquardt. Cette méthode demande de connaître les dérivées partielles par rapport à chaque paramètre du signal à optimiser pour permettre de trouver une solution. Comme nous optimisons une seule courbe delta-lognormale à la fois, nous avons été en mesure de trouver une expression analytique pour les dérivées partielles par rapport à chacun des paramètres. Ceci a permis de diminuer le temps d'extraction.

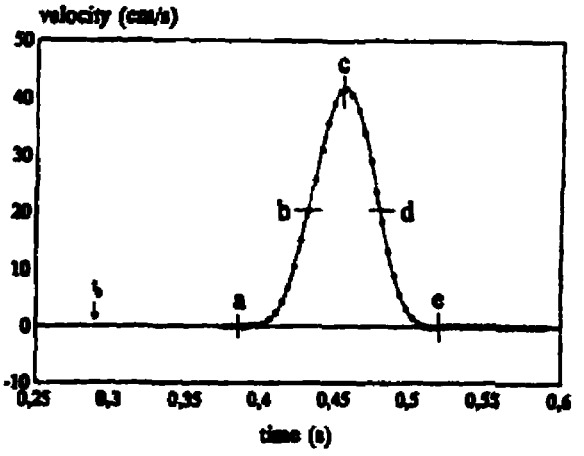


FIGURE 2.11 Segmentation d'une courbe delta-lognormale. Représentation des bornes utilisées lors de l'optimisation des paramètres des courbes delta-lognormales.

TABLEAU 2.1 Plages utilisées lors des phases d'optimisation

Phase	début	fin	paramètres
1	a	d	$D_1\ \mu_1\ \sigma_1\ t_0$
2	a	e	$D_1\ \mu_1\ \sigma_1\ t_0\ D_2\ \mu_2\ \sigma_2$
3	a	e	$D_1\ \mu_1\ \sigma_1$
4	a	e	$\mu_1\ \sigma_1$
5	a	e	$D_1\ \sigma_1\ t_0$

Lors du premier traitement, les paramètres à optimiser ainsi que la plage de la courbe utilisée varie d'une phase à l'autre. La figure 2.11 nous montre une courbe typique d'un mouvement simple ainsi que les bornes utilisées pour l'optimisation. Nous voyons par exemple, avec l'aide du tableau 2.1, que l'optimisation pour la deuxième phase se fait entre les deux minimums de vitesse. Le tableau montre aussi les paramètres qui sont optimisés. Cette procédure a été déterminée par essaies et erreurs.

Le deuxième traitement consiste à optimiser la courbure et l'angle initial. Pour ce traitement, nous avons utilisé la même procédure d'optimisation qui a été utilisée pour l'optimisation initiale, soit l'utilisation d'un tableau permettant l'optimisation simultanée. On fait deux passages, un avec des pas grossiers et l'autre avec des pas plus fins. Nous évaluons l'erreur sur la vitesse et c'est ce qui nous permet de choisir l'optimum. Comme ce calcul d'erreur est trop long si nous le faisons sur tout le signal et que, de toutes manières, pour une grande plage de valeurs, l'erreur est constante, nous utilisons une plage restreinte de valeurs pour évaluer celle-ci. Nous calculons l'erreur en utilisant la partie de la courbe delta-lognormale qui correspond à la région où nous retrouvons 98% de l'aire sous la courbe ce qui réduit de beaucoup le nombre de points.

2.4.6 Reconstruction du tracé

La dernière étape consiste à reconstruire le signal de vitesse. Le profil observé est l'addition vectorielle de tous les mouvements simples. Nous calculons donc la vitesse tangentielle à un temps donné de tous les mouvements simples et nous faisons la somme vectorielle. Comme nous avons besoin de l'aire sous la courbe pour trouver la direction de la vitesse, nous utilisons le score Z, qui permet de connaître le pourcentage cumulatif d'aire sous la courbe par rapport à la distance, en nombre d'écart type, de la moyenne, pour trouver l'aire d'une manière exacte. Ceci permet donc de réduire les erreurs cumulatives, car nous faisons la reconstruction par itération.

Dans cette section nous avons présenté un résumé de la méthode d'extraction des paramètres en deux dimensions. Nous remarquons que cette tache n'est pas facile, mais l'utilisation d'heuristiques donne de bons résultats comme nous le montrerons dans la prochaine section. Cet extracteur servira de base pour l'extracteur de paramètres à trois dimensions. Il faut cependant rappeler que la géométrie en trois dimensions est plus complexe et qu'il faudra faire plusieurs ajustements pour arriver à généraliser la méthode d'extraction.

2.5 Résultats obtenus avec le modèle 2D

2.5.1 Résultats obtenus sur des signatures

Dans la section précédente nous avons présenté un extracteur de paramètres pour le modèle delta-lognormal en deux dimensions. Son principe de fonctionnement a été résumé. Cette section présente les résultats obtenus par l'utilisation de l'extracteur de paramètres. Nous faisons un survol rapide des résultats et nous expliquons brièvement les

causes des erreurs de reconstruction. Une discussion plus complète est présentée dans le projet de fin d'études de Leduc (1998).

La figure 2.12 nous montre un cas où la reconstruction a bien fonctionné. En effet, nous voyons peu de différence entre le trait noir, qui est le tracé reconstruit, et le trait gris, qui est l'original. Même pour les meilleurs cas, nous voyons quand même des erreurs. Nous voyons par exemple que le "d" reconstruit est décalé. Il semble en effet y avoir eu une erreur sur l'évaluation de la courbure dans le bas de la lettre. Le reste de la reconstruction est cependant très bon si nous comparons visuellement la signature originale à celle reconstruite..

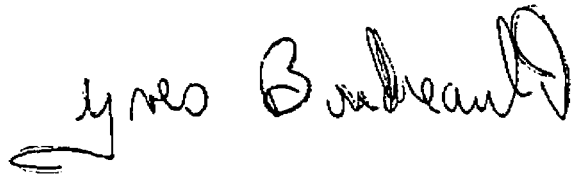


FIGURE 2.12 Exemple de reconstruction de signature.

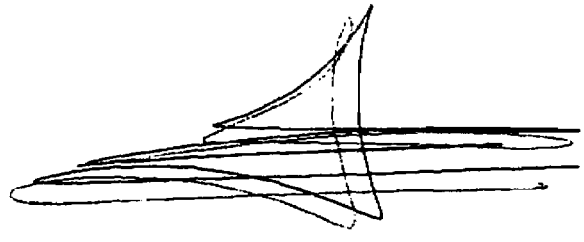


FIGURE 2.13 Reconstruction de signature décalée. Exemple d'erreur observée lors de la reconstruction de signatures.

Dans d'autres cas la reconstruction est moins bonne. On voit, par exemple dans la figure 2.13, que la signature reconstruite est décalée presque partout. Pourtant, malgré ce décalage, la forme générale semble bonne. Cette figure est un exemple de l'accumulation de l'erreur de reconstruction. On remarque qu'à certains endroits la courbure ou la longueur du trait ont été mal estimés et l'erreur locale se propage sur tout le tracé.

2.5.2 Méthode d'évaluation de l'erreur de reconstruction

Pour évaluer l'erreur, nous avons dû trouver une métrique. Nous avons choisi de calculer la longueur totale entre le tracé original et celui reconstruit. Nous avons donc calculé la distance entre les points correspondants et nous les avons additionnés. Cette mesure est cependant dépendante du mot, car plus nous avons de points, plus l'erreur est grande. Pour résoudre le problème, nous avons divisé l'erreur totale par la longueur du tracé original, ce qui nous donne une erreur relative. Le tableau 2.2 donne ce ratio pour les vingt groupes de cinq signatures analysés. Les ratios observés varient de 0,3 à 4,5 ce qui signifie que, dans le pire des cas, la somme des erreurs en terme de distance est plus de quatre fois plus grande que la longueur totale du tracé.

Il est certain que plusieurs erreurs sont gonflées à cause de la propagation de l'erreur qui peut être apparue au premier trait simple. Nous pourrions faire diminuer l'erreur substantiellement si nous déplacions le point initial du tracé reconstruit jusqu'à ce que l'erreur quadratique moyenne de la reconstruction soit minimale. En appliquant un tel algorithme, nous obtenons une erreur minimale pour les endroits où nous avons la meilleure reconstruction et non une erreur constante comme dans le cas présent. Il n'est cependant pas naturel de déplacer le point de départ d'un tracé.

Les résultats obtenus nous ont permis de voir les forces et les faiblesses de l'extracteur de paramètres. En effet, nous savons qu'il se produit souvent des erreurs sur le mouvement simple initial ce qui cause une propagation d'erreur. Nous avons remarqué que l'optimisation non-linéraire nous donne parfois des solutions complètement inacceptables. Ces erreurs sont causées par des paramètres initiaux qui ne sont pas optimaux. On voit donc, par les résultats que l'extraction est un procédé instable qui dépend du tracé original.

TABLEAU 2.2 Erreur relative pour vingt signatures

# Signature	Essai 1	Essai 2	Essai 3	Essai 4	Essai 5
1	1.976	1.552	1.598	1.154	1.322
2	1.193	0.910	0.534	0.881	0.769
3	1.800	1.959	1.521	2.305	1.443
4	1.834	1.139	0.843	1.370	1.147
5	1.678	1.538	0.862	1.275	1.597
6	0.933	0.812	1.125	0.564	1.025
7	1.174	2.427	-----	0.995	-----
8	1.829	1.017	1.445	1.593	2.395
9	1.514	1.216	0.841	1.330	0.693
10	1.190	0.939	1.345	1.323	1.379
11	0.374	0.406	4.459	0.667	0.879
12	0.685	0.757	4.437	0.892	1.255
13	1.443	1.132	1.940	1.509	1.723
14	0.912	1.462	0.773	1.430	0.818
15	2.093	1.886	1.787	1.274	1.424
16	0.363	2.045	1.066	0.492	0.656
17	0.292	0.744	0.987	0.674	0.308
18	0.848	1.134	0.490	0.647	0.604
19	1.838	1.868	2.254	0.884	0.898
20	1.790	1.723	1.659	0.729	3.713

Nous avons observé d'excellents résultats, qui ont habituellement une erreur relative de moins de 1, mais aussi des reconstructions erronées obtenues à l'aide de l'extracteur de paramètres. Nous allons maintenant présenter brièvement les causes probables de cette instabilité ainsi que les limitations de notre extracteur.

2.5.3 Cas où la reconstruction est problématique

Le plus gros problème que nous avons rencontré est que dans certains cas l'extracteur ne réussit pas à obtenir de solutions et diverge. Un cas recensé est lorsque le

profil de vitesse se termine par une accélération sans atteindre de maximum de vitesse, par exemple un signataire peut lever le crayon dans la phase d'accélération pour terminer sa signature. Dans un tel cas le programme ne trouve pas le maximum de vitesse et ceci cause un problème à cause des structures de données utilisées. Ceci a été observé dans 2% des cas analysés.

D'autres cas se présentent, mais ne sont pas aussi problématiques que le précédent. Ceux-ci causent parfois d'énormes erreurs de reconstruction. La première chose que nous faisons est la segmentation du tracé en composantes. L'utilisation des points acquis lors du contact du crayon avec la tablette n'est pas optimale, car nous nous retrouvons souvent avec des morceaux de courbes delta-lognormales. Nous pourrions garder quelques points lorsqu'il n'y a pas de contact pour compléter la courbe ce qui pourrait permettre une meilleure extraction de paramètres, car le système aurait plus de points de contrôle même si les derniers points sont plus bruités.

La détermination des maximums de vitesse à l'endroit où l'on observe des points d'inflexion cause aussi problème dans certains cas. Comme toute la méthode est basée sur l'extraction d'un nombre fixe de traits simples, il faut que le nombre de traits soit déterminé correctement. Il faudrait donc vérifier avec plus de cas la détection des maximums avec la méthode des points d'inflexion. Les paramètres choisis pour cette détection ont été déterminés par essais et erreurs et devraient faire l'objet d'une étude plus approfondie.

Pour l'estimation des courbures et des angles initiaux, plusieurs problèmes subsistent à cause de l'algorithme d'optimisation choisi. En effet, les estimations initiales servent à choisir les pas des matrices d'optimisation ce qui est limitatif. Les points d'inflexions du tracé causent aussi des problèmes. Il arrive qu'un point d'inflexion ne soit pas détecté par notre méthode ce qui cause des problèmes lors du calcul d'erreur. Celle-ci est affectée par le fait que les points doivent être placés du bon côté d'une droite déterminée

par le tracé. Enfin, il arrive aussi que le cercle ne soit pas dirigé dans le bon sens, c'est-à-dire que son centre ne soit pas à l'intérieur du tracé, à cause du bruit contenu sur le tracé. Le calcul de la courbure et de l'angle initial comporte donc beaucoup d'éléments qui occasionnent des erreurs auxquelles il faudra remédier.

Les derniers cas problématiques que nous allons présenter concernent les cibles virtuelles. Il y a plusieurs circonstances qui provoquent un mauvais positionnement de ces cibles. Dans de tels cas, nous ramenons la cible au point où nous observons la vitesse minimum. Le fait de mettre une cible au minimum de vitesse crée un trait simple qui passe à l'intérieur du tracé ce qui occasionne une courbure trop grande. Il faudrait donc revoir la correction de la cible et les règles qui permettent de décider si une cible est bonne de manière à minimiser les déplacements inutiles. Le repositionnement des cibles peut aussi créer des arcs de cercle trop grands ce qui contredit aussi nos critères de choix des cibles. Comme les cibles virtuelles permettent l'estimation de deux paramètres, il est important que leur positionnement soit le plus exact possible.

2.5.4 Modifications proposées pour passer à l'extracteur 3D

L'extracteur présenté fonctionne pour des tracés en deux dimensions malgré les quelques problèmes énoncés. Comme nous allons nous intéresser à l'extraction en trois dimensions, il faudra adapter notre extracteur. Le nouvel extracteur sera présenté en détails dans le chapitre suivant. Nous allons cependant faire une revue des points qu'il faudra adapter pour être en mesure de faire de l'extraction en trois dimensions.

Premièrement la segmentation en composantes ne sera plus nécessaire, car nous avons une acquisition continue des mouvements. Pour le cas de la recherche des maximums de vitesse, la procédure restera la même, car celle-ci ne dépend pas de la

géométrie du mouvement. Cependant, comme nous allons ajouter des paramètres au modèle il faudra ajouter des modules pour estimer et optimiser ces nouveaux paramètres.

Pour l'estimation de la courbure et de l'angle initial, il faudra étendre notre formule du cercle à la troisième dimension. De plus, il faudra porter une attention particulière aux cibles virtuelles. En effet, en deux dimensions il pouvait arriver que deux cercles ne se touchent pas, mais en trois dimensions les cercles ne se toucheront pas dans la majorité des cas. Il faudra donc trouver un moyen de trouver la meilleure cible possible. Il faudra aussi revoir les seuils et les règles permettant de vérifier les cibles.

Pour l'étape d'optimisation, celle-ci restera sensiblement la même, mais devra être étendue pour tenir compte des nouveaux paramètres. Il faudra aussi mettre à jour le calcul des dérivées partielles. La reconstruction du tracé devra aussi tenir compte des nouveaux paramètres. Il faudra surtout porter une attention particulière à la propagation probable de l'erreur, car nous allons travailler avec des vecteurs en trois dimensions.

2.6 Conclusion

Ce chapitre a présenté la théorie delta-lognormale en détails. Nous avons débuté en décrivant les origines et les fondements de ce modèle. Celui-ci a été proposé en appliquant le théorème de la limite centrale aux systèmes neuro-musculaires. Nous avons présenté le modèle en deux dimensions. Plusieurs exemples de courbes obtenues ont été montrés. L'extracteur de paramètres automatique a ensuite été expliqué. Nous avons décrit brièvement chacun des modules qui le compose et avons montré des résultats obtenus. Une brève discussion sur les problèmes et les faiblesses de l'extracteur a fait l'objet d'une section et a été suivie des points à porter attention lors de la généralisation de celui-ci. Le prochain chapitre présente en détail le modèle en trois dimensions retenu.

Chapitre 3

Théorie cinématique en trois dimensions

3.1 Introduction

Jusqu'à présent nous avons présenté différents modèles dont la plupart décrivaient des mouvements en deux dimensions. Le chapitre précédent traitait du modèle delta-lognormal en détails, parce que c'est celui-ci que nous avons choisi afin de caractériser les mouvements en trois dimensions. Dans la première section du présent chapitre, nous allons, tout de même, passer en revue les modèles présentés au premier chapitre et proposer une extension en trois dimensions pour chacun d'eux, afin de faire une critique de ces modèles généralisés et de justifier notre choix envers le modèle delta-lognormal. Par la suite, différentes avenues seront proposées pour la généralisation du modèle delta-lognormal en trois dimensions. Après avoir choisi celle qui nous semble la plus prometteuse, une méthode semi-automatique d'extraction des paramètres du modèle choisi sera décrite. Le chapitre se terminera par quelques exemples de reconstructions de mouvements.

3.2 Choix du modèle à généraliser

3.2.1 Généralisation des modèles en une dimension

Dans cette section, nous allons reprendre chacun des modèles présentés au premier chapitre et nous allons les regrouper, car plusieurs ont des caractéristiques semblables. Nous allons donc proposer une extension à trois dimensions pour chaque groupe de modèles. De plus, une courte discussion suivra chacune des extensions proposées afin

d'expliquer la raison pour laquelle des groupes de modèles n'ont pas été retenus. Cette section permettra donc de justifier notre choix d'utiliser le modèle delta-lognormal pour représenter les mouvements en trois dimensions.

Le premier groupe de modèles que nous analyserons comprend tous les modèles qui ont été présentés par Alimi (1995). En effet, ceux-ci permettent tous de reconstruire la courbe de vitesse d'un mouvement rapide. Cependant, ce sont des modèles en une dimension, car ils ne contiennent aucune information sur l'orientation du mouvement autre que le signe de la vitesse. La représentation du mouvement ne peut se faire que sur une droite dans ce cas.

En ajoutant un certain nombre de paramètres à chacun de ces modèles, il est possible de générer des tracés en deux ou en trois dimensions. En effet, il a été observé que les traces produites en deux dimensions sont des arcs de cercle. Il suffit donc d'ajouter des paramètres qui permettront de reconstruire cet arc de cercle comme la courbure et l'angle initial. Pour la troisième dimension, il faut choisir des paramètres qui caractérisent la forme des mouvements observés, par exemple, la torsion ainsi que la courbure. Il est à noter que, pour un mouvement simple, quel que soit les paramètres que nous choisissons d'ajouter pour la reconstruction de la trajectoire, ceux-ci n'ont pas d'influence sur celle du profil de vitesse. Pour représenter le groupe, nous avons donc choisi d'utiliser le modèle delta-lognormal, car c'est celui qui offre la meilleure performance pour la reconstruction des profils de vitesse d'après l'étude d'Alimi (1995).

3.2.2 Généralisation des modèles en deux dimensions

Nous pouvons regrouper tous les modèles qui sont basés sur la minimisation d'un paramètre comme le "minimum jerk" ou le "minimum torque-change". Ceux-ci demandent habituellement des calculs très complexes pour des mouvements en deux dimensions et

c'est pour cette raison que Wada et Kawato (1993) ont proposé d'utiliser un réseau neuronal pour permettre de solutionner les équations nécessitant la minimisation d'un paramètre.

Ces modèles peuvent être étendus à la troisième dimension en utilisant des paramètres vectoriels pour calculer la valeur à minimiser. Cependant, comme ceux-ci demandent déjà des calculs complexes pour des mouvements en deux dimensions, l'ajout d'une dimension supplémentaire a pour effet d'augmenter la complexité des calculs. De plus, ces modèles ont des performances qui varient selon les conditions expérimentales. Le fait de passer à la troisième dimension ne simplifie pas le problème. Il est aussi plus difficile de contrôler les conditions expérimentales, donc les résultats sont plus bruités et moins stables.

L'idée de Okadome et Masaaki (1995) de segmenter une tâche complexe en tâches élémentaires répondant au modèle "minimum jerk" aurait pu être utilisée. Par contre le fait d'utiliser le minimum jerk implique une minimisation complexe d'un paramètre. Il faut aussi rappeler que pour chaque tâche, il faut des paramètres de départ. Ce modèle demande donc trop de calculs complexes ainsi que grand nombre de paramètres dès que le mouvement est considéré complexe.

D'autres auteurs ont proposé d'utiliser des trajectoires d'équilibre ou virtuelles. L'idée de trajectoires est intéressante, car elle permet d'avoir une représentation de la manière dont le membre génère le geste réel. Le passage de la trajectoire d'équilibre en deux dimensions à la troisième dimension est assez simple. Il s'agit en fait d'ajouter une dimension aux points compris sur celle-ci.

La généralisation serait simple si seuls les points de la trajectoire étaient à déterminer. Malheureusement, pour déterminer la trajectoire d'équilibre, des modèles masses-ressorts ou musculaires sont utilisés. Pour ces modèles, le passage à la troisième

dimension est donc compliqué. Il faut prendre en compte que plusieurs joints agissants sur des axes différents sont nécessaires pour avoir des mouvements qui ne sont pas seulement planaires. Il faudra donc ajouter des muscles ou des ressorts, dépendant du modèle, pour ajouter cette nouvelle articulation. L'estimation de nouveaux paramètres est donc nécessaire et la validation de ceux que l'on connaît déjà s'avère nécessaire pour déceler les interactions probables avec ceux que l'on doit ajouter. Le nombre d'équation et de contraintes à calculer pour trouver la trajectoire augmente aussi.

Ces modèles sont rejetés, car ils demandent beaucoup trop de calculs pour la performance que nous pouvons espérer. En deux dimensions, des trajectoires virtuelles complexes étaient obtenues pour des mouvements simples. Ceci nous porte à croire qu'en trois dimensions les trajectoires seront encore plus complexes. Le modèle n'est donc pas très utile, car les trajectoires virtuelles ne sont pas une bonne représentation du mouvement observé. Le modèle utilisant les masses-ressorts est intéressant, sauf qu'il nécessite un "feed-back" constant, ce qui n'est généralement pas le cas pour des mouvements rapides.

En 1995, Goodman et Gottlieb ont proposé un modèle pour la reconstruction des mouvements en deux et en trois dimensions. Ils utilisent des équations différentielles pour produire des tracés. Ce modèle est intéressant au sens où il est possible de passer de la deuxième à la troisième dimension seulement en choisissant les bons coefficients pour les matrices contenues dans l'équation. Ce modèle a cependant pour but de relier la trajectoire cartésienne d'un geste à la trajectoire angulaire des articulations.

Dans le cadre de notre recherche, nous n'utiliserons pas ce modèle, car nous travaillons sur des mouvements simples en trois dimensions. Le modèle utilisant les équations différentielles sert surtout à trouver la position des membres pour un mouvement entre deux cibles. Ce qui nous intéresse ici est de voir comment se comporte le profil de vitesse pour des mouvements dans l'espace.

Des modèles, décrits au premier chapitre, ont été proposés pour le mouvement de la marche, par exemple, des réseaux neuronaux ou des fonctions périodiques. Les fonctions périodiques semblent adéquates si nous utilisons une suite de pas pour l'analyse. Un réseau neuronal est aussi intéressant s'il est alimenté par un neurone cyclique. Il serait facile d'ajouter une dimension à ces modèles. Pour le modèle utilisant des fonctions cycliques, il s'agirait d'ajouter un autre groupe de fonctions cycliques sur un plan qu'il faudrait déterminer. Pour le réseau neuronal, il suffit d'ajouter une sortie pour la position horizontale.

Ces modèles ne sont toutefois pas d'une grande utilité pour notre recherche. En effet, comme nous essayons de modéliser des mouvements simples, un mouvement périodique introduit de la confusion à cause des superpositions qui résultent de l'enchaînement des mouvements. De plus, le fait d'isoler un mouvement simple à partir d'une fonction périodique implique des discontinuités qui sont indésirables pour les analyses. Cependant, il est certain que l'utilisation de fonctions périodiques peut être idéale pour des gestes faits à répétition. De plus, la marche est essentiellement un mouvement en deux dimensions. L'ajout d'une troisième dimension présente moins d'intérêt pour ces modèles.

3.2.3 Critique des modèles en trois dimensions existants

Deux modèles ont été conçus pour étudier des mouvements en trois dimensions. Le premier consiste en l'utilisation de réseaux neuronaux pour reconstruire les mouvements humains. Il a été proposé par Costa et al. en 1997. Ce modèle peut être utilisé pour des applications en infographie, mais présente moins d'intérêt pour la modélisation des mouvements réels. En effet, le réseau neuronal fonctionne par itération ce qui suppose qu'il y a un "feed-back" lors de la génération d'un geste rapide. D'après la plupart des recherches effectuées dans ce domaine, nous savons que ce n'est souvent pas le cas. De

plus, avec un réseau neuronal, il est difficile d'extraire les paramètres pour chaque geste. Les paramètres existent, mais sont cachés et distribués dans les poids associés à chacun des neurones. Ils nous permettent donc difficilement de caractériser chacun des gestes. Enfin, pour qu'un réseau puisse reproduire un geste simple, il faut l'entraîner avec des centaines d'exemples. Les paramètres du réseau dépendront donc de la base d'entraînement, ce qui signifie que pour un même geste les paramètres extraits dépendront de la connaissance du réseau. Ce modèle n'est donc pas retenu, car il ne nous permet pas de trouver les paramètres qui caractérisent un geste simple facilement.

Le dernier modèle est celui de Morasso (1983). Le modèle utilise le module de la vitesse, la courbure et la torsion pour décrire les traces observées en trois dimensions. L'idée est bonne et nous l'avons réutilisée pour élaborer notre premier modèle. Morasso (1983) montre qu'à partir d'un geste il est possible de calculer trois quantités et qu'avec celles-ci, il est possible de reproduire le geste. Il ne propose aucun modèle pour expliquer les courbes observées. Par contre, il a fait une observation intéressante : pour des mouvements en trois dimensions, le maximum de courbure correspond au minimum de vitesse et vice-versa. Ceci est important, car c'est une caractéristique qui est présente pour des mouvements en deux dimensions. La torsion cause un problème, car elle ne semble pas être synchronisée avec la vitesse ou la courbure. À l'instar Morasso (1983), nous n'avons pas été en mesure de déceler une relation entre la torsion et la courbure ou la vitesse à partir des graphiques fournis dans l'article de Morasso (1983). Même s'il ne nous a pas fourni de modèle à proprement parler, ses observations sur la synchronisation entre la vitesse, la courbure et la torsion s'appliquent au modèle delta-lognormal et nous en tenons compte lors des propositions de modèles généralisés.

Nous venons donc de voir que tous les modèles en une et deux dimensions traités peuvent être généralisés à la troisième dimension. Il faut cependant rappeler que notre but n'est pas de tester chacun des modèles, mais de choisir celui qui est le plus susceptible

d'avoir la meilleure performance pour la reconstruction de trajectoires en trois dimensions. Dans cette perspective, nous avons choisi le modèle delta-lognormal, car celui-ci permet, mieux que tout autre modèle, de reconstruire les profils de vitesse ainsi que les tracés des mouvements en une et deux dimensions. Cette conclusion est basée sur la comparaison de l'erreur quadratique moyenne de reconstruction.

3.3 Propositions d'un modèle généralisé

Dans la section précédente nous avons proposé des extensions aux modèles en une et deux dimensions trouvés dans la littérature. Nous avons choisi le modèle à généraliser et nous avons donné les raisons de ce choix. Nous allons maintenant présenter les propositions qui ont été analysées, mais rejetées pour l'extension du modèle delta-lognormal en trois dimensions. Cette section présente les propositions, les fondements de celles-ci ainsi que les raisons qui nous ont poussés à les rejeter. Nous présentons donc ici, dans l'ordre, un modèle à torsion variable et un modèle à torsion constante.

3.3.1 Théorie du trièdre de Frenet

Avant de présenter les modèles, nous allons tout d'abord rappeler ce qu'est le trièdre de Frenet. Imaginons une courbe paramétrique en trois dimensions où les positions en x, y et z dépendent du temps. Nous pouvons trouver un point sur cette courbe en sélectionnant le paramètre temporel qui lui correspond. À chaque point, nous pouvons associer un ensemble de trois vecteurs orthogonaux et de trois plans qui forment le trièdre de Frenet. Celui-ci est représenté schématiquement à la figure 3.1.

Le premier plan, qui est utilisé pour caractériser un point, est celui qui contiendrait la courbe si aucune torsion n'était présente en ce point. Deux vecteurs sont contenus sur ce plan, soit le vecteur tangent et le vecteur normal. Le vecteur tangent ($\vec{t}$) est, comme son

nom l'indique, tangent à la courbe et pointe dans la direction du déplacement d'une particule qui suivrait la courbe. Le vecteur normal ($\vec{n}$) pointe dans le sens concave de la courbe et est perpendiculaire au vecteur tangent. Le vecteur binormal ($\vec{b}$) est défini simplement par le produit vectoriel $\vec{T} \times \vec{n}$. Comme ces trois vecteurs sont, par définition, unitaires et perpendiculaires l'un à l'autre, le référentiel formé par ceux-ci est orthonormé. En plus, deux autres plans sont définis par Frenet. Il y a le plan normal dont le vecteur normal est $\vec{T}$ et le plan de rectification dont $\vec{n}$ est sa normale.

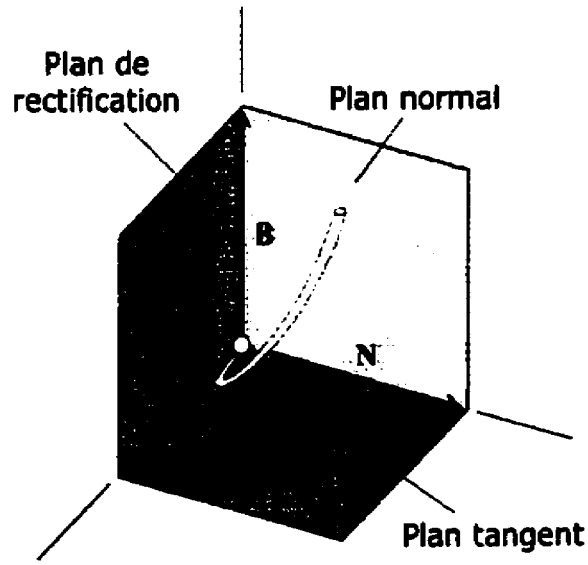


FIGURE 3.1 Représentation du trièdre de Frenet.

Comme ce qui nous intéresse est le module de la vitesse, la courbure et la torsion, nous allons utiliser le trièdre de Frenet pour définir chaque terme géométriquement. Le module de la vitesse est défini comme étant le déplacement du trièdre dans la direction du

vecteur tangent. La courbure, pour sa part, représente le changement de direction de $\vec{T}$ autour de $\vec{B}$, c'est à dire sur le plan contenant la courbe s'il n'y avait pas de torsion. L'inverse de la courbure est le rayon du cercle tangent à la courbe. Le vecteur normal pointe dans la direction du centre de ce cercle. Le vecteur tangent tournera autour de ce cercle ainsi que $\vec{N}$, car $\vec{T}$ et $\vec{N}$ doivent être perpendiculaires.

La torsion peut être définie comme la rotation du trièdre autour du vecteur tangent. Le vecteur normal se déplace aussi sur un cercle qui a un rayon égal à l'inverse de la torsion et dont le centre est dans la direction pointée par le vecteur binormal. La torsion est, dans ce contexte, une extension de la courbure pour la troisième dimension. La connaissance du module de la vitesse, de la courbure et de la torsion nous permet donc de reconstruire une courbe si nous connaissons le trièdre ainsi que le point initial.

$$\vec{t}=\frac{\vec{r}}{|\vec{r}|} \quad \vec{b}=\frac{\vec{r}\,x\dot{\vec{r}}}{|\vec{r}\,x\dot{\vec{r}}|} \quad \vec{n}=\vec{b}x\vec{t}$$
$$|\vec{v}|=|\vec{r}| \quad \kappa=\frac{|\vec{r}\,x\dot{\vec{r}}|}{|\vec{r}|^3} \quad \tau=\frac{(\vec{r}\,x\dot{\vec{r}}).\ddot{\vec{r}}}{|\vec{r}\,x\dot{\vec{r}}|^2} \tag{3.1}$$

Les équations (3.1) permettent de calculer le module de la vitesse ($\vec{v}$), la courbure (κ) et la torsion (τ) ainsi que les vecteurs $\vec{T}$, $\vec{N}$ et $\vec{B}$. Il est à noter que $\dot{r}$ signifie que l'on doit utiliser la première dérivée du vecteur position par rapport au temps, $\ddot{r}$, la seconde dérivée et $\dddot{r}$, la troisième dérivée.

3.3.2 Première proposition : modèle à torsion variable

Maintenant que les différents paramètres que nous allons utiliser sont définis, nous pouvons passer au premier modèle proposé. Le but est d'étendre le modèle delta-lognormal à la troisième dimension. Nous savons que sa version en une dimension permet de décrire l'évolution du module du vecteur vitesse en fonction du temps (Plamondon, 1995a). Pour

être en mesure de reproduire des tracés en deux dimensions la courbure ainsi qu'un angle initial ont été ajoutés (Plamondon et Guerfali, 1998).

La proposition naturelle pour généraliser le modèle serait d'y ajouter la torsion. Nous avons analysé un modèle à torsion variable comportant 15 paramètres, soit 7 paramètres pour le module de la vitesse de forme delta-lognormale, la courbure, la torsion et 6 paramètres pour définir deux vecteurs du trièdre de Frenet, soit les vecteurs tangent et normal. À l'aide de ces derniers, il nous est possible de retrouver le vecteur binormal ainsi que les trois plans. Ce modèle permet, en théorie, de reconstruire n'importe quel mouvement dont nous connaissons les paramètres.

Ce modèle-ci, comme tous les autres, est basé sur la définition du mouvement simple à savoir un mouvement dont on observe un seul profil delta-lognormal sur le graphique de vitesse. Ceci n'implique rien en ce qui concerne la courbure ou la torsion. Nous pouvons donc nous attendre à ce que ces paramètres ne soient pas constants. Dans ce cas, il faudra trouver des expressions mathématiques pour définir l'évolution de ces paramètres.

La figure 3.2 tiré d'un article de Morasso (1983), nous montre un exemple de courbure et de torsion pour une série de mouvements en trois dimensions. Nous remarquons que la courbure est déphasée de 90 degrés par rapport au module de la vitesse et semble être constante pour des mouvements simples malgré la superposition des mouvements qui rend l'analyse plus difficile. Cette observation avait déjà été faite pour des mouvements en deux dimensions, ce qui laisse croire que l'hypothèse serait bonne. Pour la torsion il est plus difficile de trouver une tendance. Pour l'instant nous assumons qu'elle est définie par une fonction générale jusqu'à ce que nous soyons en mesure de mieux la préciser.

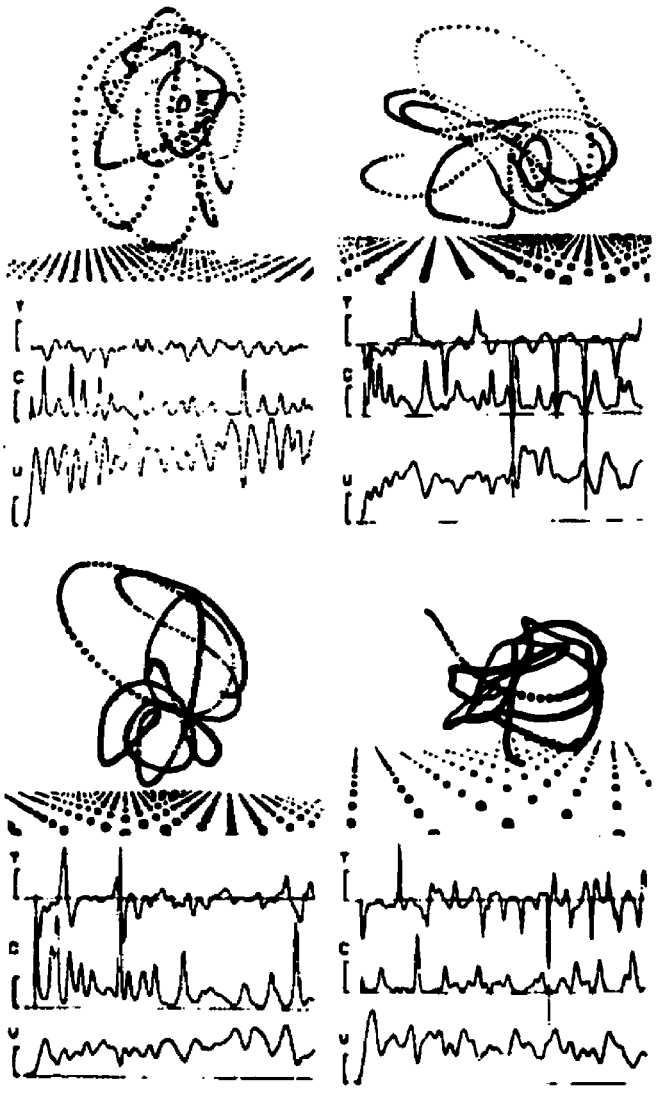


FIGURE 3.2 Exemples de profils de vitesse, courbure et torsion. Pour chacun des quatre mouvements complexes, on voit en dessous les profils du module de la vitesse, de courbure et de torsion. (Source : Morasso, 1983, Fig 4)

En supposant que nous connaissions tous les paramètres, nous nous sommes concentrés sur la reconstruction de la courbe. En effet, il faut avoir un moyen pour simuler ce nouveau modèle à torsion variable et reconstruire les mouvements à l'aide des paramètres que nous prévoyons extraire. Nous avons donc travaillé sur le calcul de la trajectoire à partir des équations (3.2) de Frenet :

$$\frac{d\vec{t}}{dt}=\kappa|\dot{r}|\vec{n} \quad \frac{d\vec{n}}{dt}=-\kappa|\dot{r}|\vec{t}+\tau|\dot{r}|\vec{b} \quad \frac{d\vec{b}}{dt}=-\tau|\dot{r}|\vec{n} \tag{3.2}$$

D'après l'article de Morasso (1983), il est possible de retrouver les positions en fonction du temps en intégrant les formules précédentes. Ces calculs s'avérant complexes, nous avons décidé d'utiliser les propriétés géométriques de la courbure et de la torsion pour arriver à nos fins. Il s'agissait donc de faire tourner le vecteur tangent ainsi que le vecteur normal dans la direction calculée à l'aide de la courbure et de la torsion pour savoir dans quelle direction déplacer le trièdre pour être en mesure de trouver itérativement la position de chaque point du tracé.

Cette méthode fonctionne bien à certaines conditions. En effet, il y a des cas extrêmes qu'il faut éviter, car, si le nombre de points est trop petit, la reconstruction devient alors très mauvaise qualité ou, si le nombre de points est trop grand, elle demande un temps de calcul très long. Un problème est présent si nous n'utilisons pas assez de points ou que la fréquence d'échantillonnage est trop faible. Dans de tels cas, comme nous proposons de calculer le changement de direction dans chacun des axes une seule fois, la direction du vecteur tangent aura tendance à s'éloigner de sa véritable direction. Dans le cas idéal, la rotation autour de $\vec{t}$ et de $\vec{b}$ devrait se faire simultanément ce qui est donc loin d'être le cas lors dans notre logiciel.

Pour permettre de réduire au minimum l'erreur de reconstruction causée par un nombre de points insuffisant, nous avons donc décidé de subdiviser l'intervalle entre deux

points en sous intervalles. De plus, l'ordre de rotation est inversé pour chaque calcul. En appliquant cette théorie, l'erreur de reconstruction diminue de beaucoup. Cependant, si nous ne faisons pas attention, nous risquons d'arriver à l'autre extrême, c'est à dire, un temps de calcul qui devient très long. Il faut donc trouver un juste milieu pour que l'algorithme de génération de tracé puisse être utilisé pour la reconstruction.

Après quelques tests sur une spirale dont la courbure et la torsion étaient connus, il nous est apparu que pour avoir une erreur acceptable, il suffisait de diviser l'intervalle temporel entre deux points en deux sous intervalles. Cependant, pour un mouvement, le nombre de sous intervalles nécessaire pourrait être différent. Il faudra donc en tenir compte lorsque nous utiliserons cette méthode, à la fois pour générer les courbes et calculer les erreurs de reconstruction.

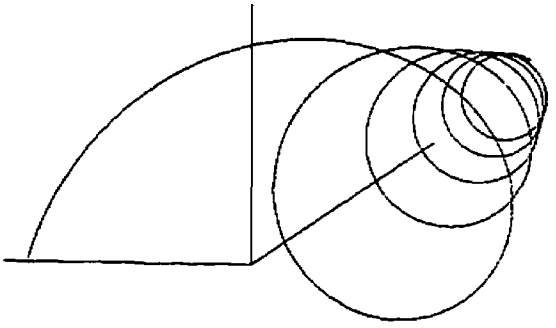


FIGURE 3.3 Spirale reconstruite.

Nous avons développé et implanté un programme de reconstruction basé sur le modèle à torsion variable. Après l'avoir testé et validé, nous nous sommes attaqués à l'extraction des paramètres. Les équations théoriques nous permettent, dans des cas idéaux, de calculer la courbure et la torsion. Malheureusement, à cause du bruit contenu dans les signaux, nous avons obtenu des résultats catastrophiques. Ces équations sont très sensibles au bruit, car elles contiennent des dérivés multiples. Comme nous le savons, chaque

dérivée amplifie le bruit d'un signal. Le rapport signal sur bruit se détériore et il devient vite impossible de traiter les signaux.

Pour tenter de résoudre le problème, nous avons filtré le signal. Cependant, le moment où l'on applique les filtres ainsi que la fréquence de coupure ont une grande importance. Après quelques essais, nous en sommes venus à la conclusion que le signal de la torsion calculé ne permettait pas de bien décrire la composante de torsion d'une trajectoire, car la formule pour son calcul est beaucoup trop instable, car il il fait intervenir de dérivés d'ordre trois et il dépend de beaucoup de paramètres.

3.3.3 Seconde proposition : modèle à torsion constante

Comme nous n'avons pas été en mesure de trouver une fonction pouvant décrire la torsion d'un mouvement simple, nous avons revu notre approche et avons développé un modèle à torsion constante où nous utilisons les mêmes paramètres que pour le premier, mais en proposant une torsion constante pour un mouvement simple. La définition de ce dernier, d'après ce nouveau modèle, est un mouvement dont la vitesse a un profil de forme delta-lognormal et dont la courbure et la torsion sont constantes.

À première vue, cette nouvelle proposition correspond mieux aux gestes observés. En effet, il faut générer des mouvements faisant intervenir plusieurs articulations pour que la torsion apparaisse. Celle-ci provoque la rotation des axes du trièdre de Frenet. Cette rotation fait sortir le tracé du plan le contenant localement. Donc, pour qu'il y ait de la torsion, il faut que deux mouvements simples soient superposés de manière temporelle et soient exécutés dans des plans différents.

Le fait que plusieurs groupes musculaires agissant sur des articulations différentes doivent être sollicités, a pour effet de générer des profils de vitesse avec plusieurs pics. Il

est difficile d'activer deux groupes musculaires simultanément de manière à ce que deux courbes delta-lognormales soient complètement superposées et qu'il soit impossible de distinguer les deux mouvements simples. Cependant, dans certains cas, le problème se pose et c'est pour cela qu'une torsion constante est utilisée.

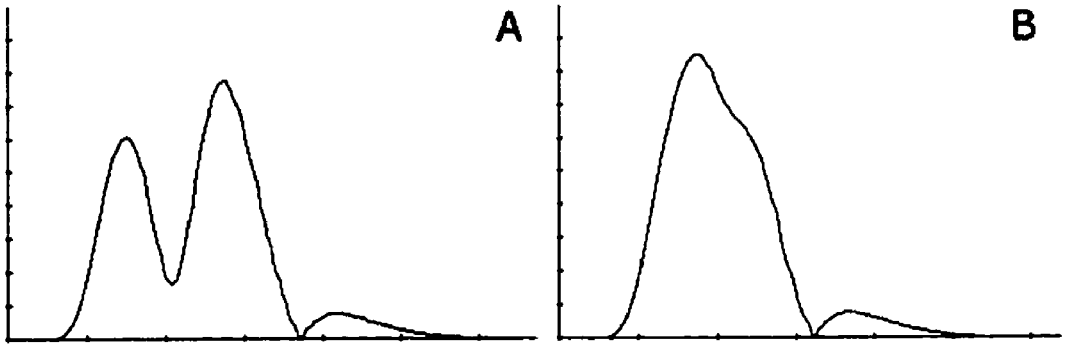


FIGURE 3.4 Exemple de superposition. Dans le cas A, il y a une légère superposition des mouvements simples. Pour le cas B, la superposition est complète et nous ne sommes pas en mesure de distinguer les deux mouvements simples.

L'algorithme de reconstruction présenté pour le modèle à torsion variable s'applique aussi pour le modèle à torsion constante. En effet, nous utilisons les mêmes paramètres pour reconstruire les profils de vitesse, courbure et torsion de même que pour reconstruire les gestes. On peut aussi s'attendre à une meilleure reconstruction, car les paramètres sont connus exactement et l'évaluation de la courbure est plus facile, car nous savons qu'elle est constante. Dans le modèle à torsion variable, nous ne connaissions pas la fonction exacte pour la torsion ce qui aurait produit des erreurs cumulatives lors de la reconstruction.

Par contre, le même problème que dans le modèle à torsion variable est présent, c'est-à-dire le calcul des paramètres. Comme nous utilisons toujours les mêmes formules, le problème d'amplification du bruit est toujours présent. Cependant, comme nous savons

que la torsion doit être égale à zéro dans beaucoup de cas et être, dans de rares conditions, constante, nous pouvons nous permettre d'utiliser des approximations plus grossières.

Après l'avoir implanté et testé, ce modèle à torsion constante n'a pas été retenu, car nous nous sommes aperçus qu'il ne correspondait pas à nos attentes quant aux performances pour la reconstruction de mouvements simples. Nous recherchons une manière de généraliser le modèle delta-lognormal, mais il ne faut pas oublier qu'il est basé sur des observations physiologiques et psychophysiques concernant la méthode de génération d'un geste par les systèmes neuro-musculaires. Les deux extensions présentées ici sont basées sur des propriétés mathématiques exclusivement. En effet, nous avons défini le geste simple comme étant composé d'un profil de vitesse ayant une seule courbe delta-lognormale. Dans la suite de nos travaux, nous avons donc cherché une autre manière de le généraliser en tenant compte et exploitant le travail déjà fait.

3.4 Modèle delta-lognormal généralisé

Deux extensions pour le modèle delta-lognormal ont été proposées à la section précédente, mais aucune n'a été retenue, car elles ne représentaient pas bien les mouvements observés. Cette section présente le modèle retenu ainsi que les raisons qui ont motivé le choix des paramètres. Une explication concernant l'émergence de la torsion est aussi proposée.

3.4.1 Définition du geste simple en trois dimensions

Comme les modèles précédents avaient une définition du geste simple qui était mathématique, nous avons dû revoir celle-ci. Le modèle delta-lognormal s'appuie à la base sur la propagation de l'influx nerveux du cerveau jusqu'aux muscles agonistes et antagonistes. Une paire d'impulsions synchrones permet d'activer une paire de systèmes

neuro-musculaires et donc de provoquer le déplacement d'un membre. Notre définition pour le geste simple en trois dimensions est basée sur ce principe. Elle peut être exprimée comme suit : un geste simple est tout geste qui n'a nécessité qu'une seule paire d'impulsions synchrones provenant du cerveau, donc qui ne fait intervenir qu'une seule synergie constituée de deux systèmes neuro-musculaires, un agoniste et l'autre antagoniste agissant dans un seul plan.

Cette définition décrit clairement les gestes qui peuvent être considérés comme simples. Il est important de se rappeler qu'un muscle ne peut que se contracter ou se relâcher. Il n'est donc pas possible de faire un mouvement dans deux plans différents avec un seul muscle. Lorsqu'une impulsion active un système neuro-musculaire, plusieurs muscles peuvent être sollicités, mais certains auront une influence plus importante sur le mouvement généré. Comme ces derniers sont généralement peu nombreux, nous observerons un mouvement se produisant dans un seul plan ou la rotation d'un membre autour d'une articulation. Nous généralisons donc cette approche en l'appliquant à une synergie agissant dans un seul plan, définissant ainsi le geste simple. Les mouvements complexes résultent donc d'une activation séquentielle de plusieurs synergies agissant éventuellement dans des plans différents.

3.4.2 Modèle delta-lognormal généralisé

Ces considérations nous ont amené à proposer un modèle où chaque mouvement simple est caractérisé par onze paramètres. Les sept premiers sont D_1, μ_1, σ_1, t_0, D_2, μ_2 et σ_2 qui permettent de décrire le module du vecteur vitesse à l'aide de l'équation delta-lognormale. Les deux suivants sont C_0 qui représente la courbure du geste et θ_0 qui est son orientation initiale dans le plan XY. À ces neuf paramètres du modèle en deux dimensions, nous ajoutons ϕ et ρ qui permettent de définir l'orientation du plan contenant le geste dans l'espace. Le paramètre ϕ représente la rotation autour de l'axe X tandis que ρ est l'angle

de rotation autour de Z. Les équations suivantes permettent de calculer la vitesse vectorielle pour un geste simple en un point.

$$|\vec{v}(t)| = \vec{D}_{1_{(P_0,\theta_0,C_0,\rho_0,\phi_0)}} \Lambda(t;t_0,\mu_1,\sigma_1^2) - \vec{D}_{2_{(P_0,\theta_0,C_0,\rho_0,\phi_0)}} \Lambda(t;t_0,\mu_2,\sigma_2^2)$$

$$\text{où} \qquad \Lambda(t;t_0,\mu_i,\sigma_i^2) = \frac{1}{\sigma_i\sqrt{2\pi}(t-t_0)} e^{\frac{-[\ln(t-t_0)-\mu_i]^2}{2\sigma_i^2}}$$

$$\theta(t) = \theta_0 + C_0 \int_{t_0}^{t} |\vec{v}(\tau)| d\tau \qquad (3.3)$$

ainsi:

$$v_x = |\vec{v}(t)| * \cos(\theta(t)) * \cos(\rho_0) - |\vec{v}(t)| * \sin(\theta(t)) * \cos(\phi_0) * \sin(\rho_0)$$
$$v_y = |\vec{v}(t)| * \cos(\theta(t)) * \sin(\rho_0) + |\vec{v}(t)| * \sin(\theta(t)) * \cos(\phi_0) * \cos(\rho_0)$$
$$v_z = |\vec{v}(t)| * \sin(\theta(t)) * \sin(\phi_0)$$

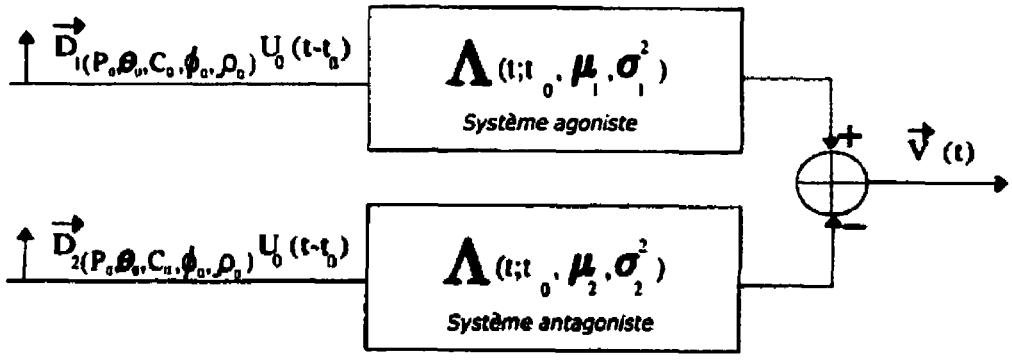


FIGURE 3.5 Synergie de génération des mouvements en 3D. Le schéma montre deux systèmes neuromusculaires formant une synergie. Le profil de vitesse observé lors d'un mouvement est la réponse impulsionnelle de ce système.

Nous tenons à souligner que les équations permettant de calculer v_x, v_y et v_z proviennent d'une multiplication de matrices de rotation. Tout d'abord, nous avons fait une

rotation autour de l'axe X et ensuite autour de l'axe Z. Il faut faire attention de ne pas inverser l'ordre des rotations si nous voulons faire le calcul une rotation à la fois. En effet, lorsque l'on calcule celles-ci, les calculs ne peuvent pas être permutés. Si nous prenons par exemple le vecteur (1,0,0) et que nous lui faisons faire une rotation de 90 degrés autour de l'axe Z suivi d'une de 90 degrés autour de l'axe X nous obtiendrons le vecteur (0,0,1). Si nous faisons les rotations dans le sens inverse, nous aurons dans ce cas (0,1,0). La figure 3.6 illustre ce qui se produit.

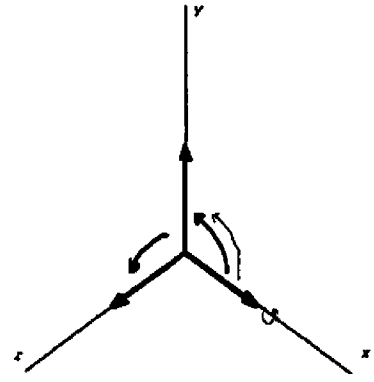


FIGURE 3.6 Propriété des rotations. Les traits minces correspondent à deux rotations faites de manière séquentielle: la première autour de l'axe X et la seconde autour de l'axe Z. Pour les traits épais, l'ordre de rotation a été inversé.

3.4.3 Implications reliées au modèle

Pour le cas des mouvements complexes, nous généralisons l'approche proposée pour étudier les mouvements à deux dimensions et nous utilisons la théorie des cibles virtuelles, qui est illustrée à la figure 3.7, pour les expliquer. Le cerveau, pour exécuter un mouvement, se fixe une cible virtuelle pour estimer la paire d'impulsions à envoyer aux systèmes neuro-musculaires pour atteindre son but. Si, en cours de chemin, il décide d'atteindre une nouvelle cible, le cerveau sait que le membre va atteindre la première cible

virtuelle visée dans un certain temps et avec une certaine précision déterminée par le ratio D_1/D_2. Sachant cela, il donnera une commande pour relier la première cible qu'il atteindra à la seconde cible. Le trajet complexe que nous allons observer dépendra de la position des cibles ainsi que du moment d'application de la paire de commande advenant un nouveau mouvement simple dans un autre plan.

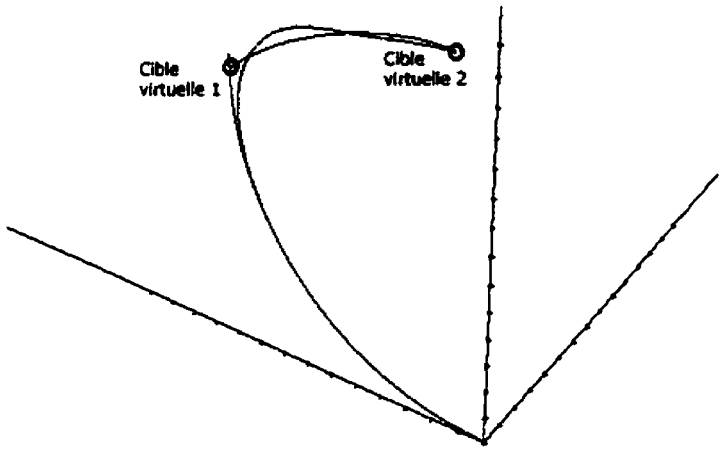


FIGURE 3.7 Concept de cibles virtuelles. Une personne fait un mouvement vers la première cible virtuelle, mais active un deuxième mouvement avant de l'atteindre ce qui génère une trajectoire complexe.

En activant successivement plusieurs mouvements simples, il sera donc possible de générer des trajectoires complexes. De plus, le fait que chacun de ces mouvements soit exécuté dans des plans différents, nous voyons apparaître automatiquement la torsion. Cette dernière n'a donc pas a être incluse dans notre modèle comme une propriété contrôlée, car elle apparaît naturellement. De plus, il est normal qu'il n'y ait pas de synchronisation entre les graphiques de vitesse et de torsion d'après notre modèle. En effet, la torsion apparaît lorsque deux gestes simples se superposent partiellement dans le temps. Elle dépend de l'orientation de chacun des plans d'exécution. Elle sera donc présente à chaque fois qu'il y aura superposition de mouvements simples, à moins que ceux-ci soient dans des plans

parallèles. La position des maximums de torsion dépendra de plusieurs facteurs, dont le degré de superposition ou de l'orientation relative des plans.

La figure 3.8, montre les courbes de vitesse, de courbure et de torsion obtenues par Morasso (1983) pour une trajectoire complexe. Celle-ci semble confirmer ce que prédit notre modèle, même si la superposition des mouvements simples est grande et rend l'analyse des courbes légèrement difficile. En effet, nous observons une synchronisation entre les graphiques de courbure et de vitesse, mais la torsion n'est pas synchronisée avec ces dernières. Même si la qualité de l'image de la trajectoire n'est pas très bonne, nous remarquons que la courbure est minimale et est presque constante aux maximums de vitesse et que les changements d'orientation de la trajectoire, provoquant la torsion, arrivent à différents moments.

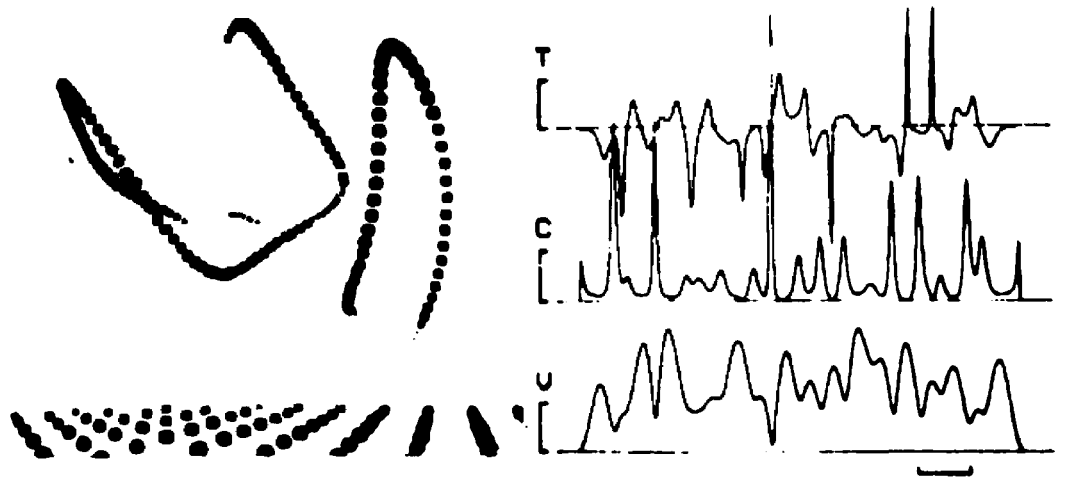


FIGURE 3.8 Profils de vitesse, courbure et torsion. A gauche on voit une trajectoire en trois dimensions et à droite les profils de vitesse (V), courbure (C) et torsion (T) associés. (Source : Morasso, 1983, Fig. 6)

La figure 3.9, produite par Morasso (1983), présente la distribution des pics de courbure et de torsion par rapport au moment où ils se produisent entre deux maximums de vitesse sur une échelle de position relative. Ce qu'il faut remarquer c'est que la distribution

des pics de torsion est presque uniforme tandis que ceux de courbure semblent suivre grossièrement une loi normale. Ceci confirme, une fois de plus, ce que prédit notre modèle, soit que la torsion est une conséquence de la superposition de gestes simples. La figure 3.9 démontre bien qu'il y a une synchronisation entre la courbure et la vitesse, car la position relative des pics de courbure est regroupée. La torsion n'est pas synchronisée avec la vitesse, ce qui provoque une distribution uniforme de la position des pics de torsion observés.

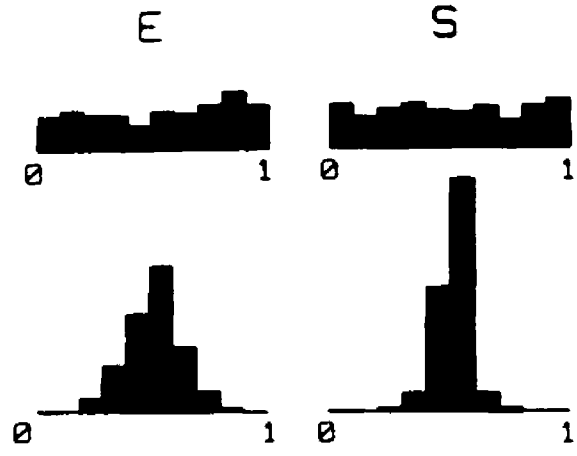


FIGURE 3.9 Délai d'apparition de la courbure et de la torsion. Histogrammes représentant la distribution du délai d'apparition de la courbure (bas) et de la torsion (haut). Ces résultats proviennent d'expérimentations (E) ou sont simulés (S). (Source : Morasso, 1983, Fig.5)

Le modèle delta-lognormal en trois dimensions comprends donc onze paramètres par mouvement simple. Il prend en compte les aspects physiologiques de la génération des mouvements. Il ne considère pas la torsion comme un facteur de contrôle, car il permet d'expliquer sa présence pour des gestes complexes par simple conséquence de l'addition vectorielle de deux vecteurs vitesse. De plus, le concept de cibles virtuelles utilisé en deux dimensions est repris intégralement ici. Dans le reste du chapitre, une méthode pour

extraire les paramètres est proposée et des simulations du modèle sont présentées pour des gestes simples et complexes.

3.5 Méthode d'extraction de paramètres proposée

La section précédente a été consacrée au modèle delta-lognormal en trois dimensions. Nous avons traité de ses fondements ainsi que de ses implications. Pour être en mesure de tester et mesurer les performances du modèle il faut extraire les paramètres de celui-ci. Cette section est consacrée à la présentation d'un extracteur, qui a été utilisé dans ce projet, pour les mouvements simples. Les points à prendre en compte lors de l'adaptation de celui-ci pour les mouvements complexes sont aussi discutés.

3.5.1 Extracteur utilisé pour les mouvements simples

Premièrement, voyons comment nous allons extraire les paramètres pour un mouvement simple. Le cas est assez facile à traiter, car la reconstruction en trois dimensions et le profil de vitesse peuvent être traités indépendamment. En effet, la modification de la courbure ou de l'orientation du plan contenant le geste n'a pas d'influence sur l'erreur de reconstruction du profil de vitesse dans le cas d'un geste simple, car aucune addition vectorielle de la vitesse n'est présente. Pour un geste complexe, c'est différent, car il faut tenir compte des additions vectorielles lorsque nous calculons le profil de vitesse reconstruit. L'erreur sur la vitesse est donc influencée par la superposition vectorielle de gestes simples.

Le choix de la courbure et du plan n'influe donc pas sur le profil de vitesse, mais l'inverse n'est pas vrai. Si nous commettons une erreur sur le profil de vitesse, celle-ci aura un effet sur la reconstruction en trois dimensions, car la distance entre chaque point dépend

de la vitesse. Ceci nous amène à conclure que l'extraction des paramètres du vecteur vitesse est très importante, car elle a un effet sur les deux reconstructions.

La méthode proposée pour extraire les sept paramètres de la courbe delta-lognormale utilise l'estimation graphique suivie de l'optimisation non-linéaire à l'aide de la méthode Levenberg-Marquardt. Cette méthode a été utilisée pour des tracés en une et deux dimensions et donne de bons résultats.

Pour l'estimation des paramètres définissant le plan du geste ainsi que la courbure et l'orientation initiale de celui-ci, l'extraction est un peu plus complexe. En effet, l'optimisation non-linéaire peut être utilisée, mais il faut déterminer les paramètres initiaux. La méthode la plus simple est de commencer par trouver le meilleur plan contenant les points tri-dimensionnels. Il faut donc déterminer le vecteur perpendiculaire au plan. Pour ce faire, nous devons minimiser la somme des distances des points au plan. En calculant les dérivées partielles de la distance totale par rapport à chaque paramètre, il est possible de résoudre un système d'équations (3.4) pour retrouver le vecteur recherché.

$$\begin{gathered} a*\Sigma[x(i)^2]+b*\Sigma[x(i)*y(i)]+c*\Sigma[x(i)*z(i)]=-d*\Sigma[x(i)] \\ a*\Sigma[x(i)*y(i)]+b*\Sigma[y(i)^2]+c*\Sigma[y(i)*z(i)]=-d*\Sigma[y(i)] \\ a*\Sigma[x(i)*z(i)]+b*\Sigma[y(i)*z(i)]+c*\Sigma[z(i)^2]=-d*\Sigma[z(i)] \\ a*\Sigma[x(i)]+b*\Sigma[y(i)]+c*\Sigma[z(i)]=-d*n \\ a^2+b^2+c^2=1 \end{gathered} \tag{3.4}$$

Dans le système d'équations (3.4), les paramètres a, b et c son ceux définissant le vecteur perpendiculaire au plan. Le système a une solution unique seulement si tous les points font parti du même plan. Dans tous les autres cas le système n'a pas de solutions, car il y a plus de contraintes que d'inconnus. L'implantation n'a été d'aucune utilité, car nous arrivions toujours à la conclusion que le système n'avait pas de solution. Ce qu'il aurait fallu faire est d'implanter une méthode numérique utilisant le critère des moindres carrées pour résoudre le système de manière à trouver une erreur minimale. Comme ceci

dépassait le cadre de ce projet de maîtrise, nous nous sommes donc contentés à extraire les paramètres manuellement à l'aide d'un logiciel programmé dans ce but précis.

Si nous implantons éventuellement un algorithme pour calculer l'orientation du plan, il faudrait ensuite ramener tous les points sur le plan XY. Une fois que cette transformation est réalisée, nous nous retrouvons dans le cas d'un tracé en deux dimensions. Il est donc possible d'utiliser les méthodes d'extraction en deux dimensions pour retrouver la courbure et l'angle initial.

3.5.2 Extracteur pour les mouvements complexes

La méthode qui a été présentée fonctionne pour un geste simple. Cependant, les gestes que nous faisons sont habituellement complexes. L'extracteur ne pourrait donc pas être utilisé directement. Tout comme en deux dimensions, la superposition vectorielle se produit sur le profil de vitesse. Pour être en mesure de pouvoir extraire les paramètres, il faudrait idéalement travailler sur un seul profil de vitesse à la fois. Nous pouvons donc utiliser la méthode présentée au deuxième chapitre qui consiste à soustraire tous les profils déjà connus pour ne garder que celui sur lequel nous voulons faire de l'extraction ou de l'optimisation.

Un autre problème est présent lors de l'estimation de la courbure et de l'orientation du plan. En effet, un geste complexe n'est habituellement pas compris dans un seul plan, donc il faut trouver une stratégie pour déterminer l'orientation du plan ainsi que la courbure de chaque geste simple. Un moyen proposé serait, comme pour l'extraction en deux dimensions, d'utiliser les points près de chaque maximum de vitesse, car ceux-ci sont habituellement moins affectés par la superposition. Nous croyons que cette stratégie devrait donner une bonne estimation initiale des paramètres qui seront par la suite optimisés. Comme nous n'avons pas implanté l'extracteur automatique de paramètres, ceci reste une

hypothèse, mais nous croyons qu'elle devrait être prise en compte lors de la conception de celui-ci.

Il est donc assez simple de faire un extracteur pour des mouvements simples, mais le cas se complique pour des mouvements complexes à cause de la superposition vectorielle. Lors du prochain chapitre, nous expliquerons les méthodes utilisées pour obtenir les résultats présentés, et celles-ci comprennent l'extracteur qui a été utilisé.

3.6 Exemples de reconstruction

La section précédente présente un extracteur pour les paramètres du modèle delta-lognormal généralisé en trois dimensions. Il décrit l'extracteur utilisé pour les mouvements simples et en propose un pour des mouvements complexes. Dans cette section nous avons utilisé le module de reconstruction des trajectoires de l'extracteur pour simuler des mouvements simples et complexes. Les résultats obtenus pour la reconstruction de gestes réels sont présentés au chapitre suivant.

3.6.1 Mouvements simples

Pour terminer ce chapitre, nous allons présenter quelques trajectoires en trois dimensions simulées ainsi que les profils de vitesse correspondants. Pour plus de clarté, nous n'avons pas mis de personnages montrant les positions de départ et finales des mouvements sur la figure 3.10. Celle-ci ne montre qu'une ligne en trois dimensions qui représente la position de l'extrémité du membre simulé. Dans toutes les simulations, la position de départ est le point (0,0,0) ce qui ne permet pas de différencier les gestes par leur position dans l'espace. Cependant la forme de la trajectoire n'est pas modifiée par cette translation.

La figure 3.10 montre quelques mouvements simples dans plusieurs axes. On voit les courbes de vitesse qui ont servi à générer les traces en trois dimensions. Les traces présentées pourraient par exemple être celles d'un bras. On voit que le modèle permet de générer des mouvements d'amplitudes et de vitesses différentes. De plus la direction de la trajectoire peut être ajustée, pour tenir compte du plan dans lequel s'est effectué le geste.

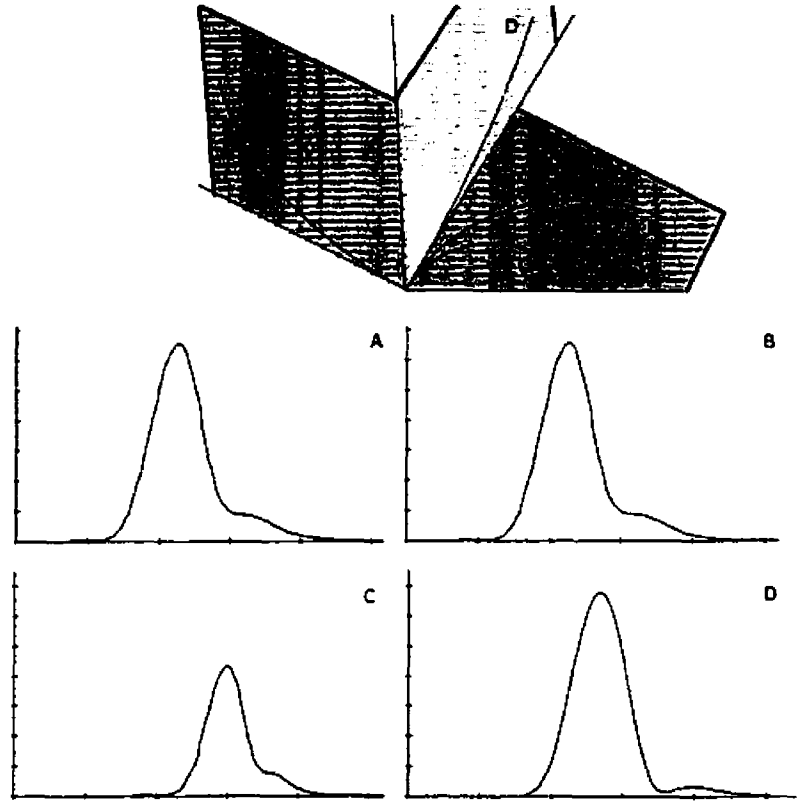


FIGURE 3.10 Exemple de mouvements simples. Quatre exemples de mouvements simples dans différents plans.

3.6.2 Mouvements complexes

Une famille de mouvements complexes est illustrée à la figure 3.11. Nous voyons plusieurs trajectoires qui ont permis de générer la même séquence de mouvements. Cette

séquence aurait pu être le lever du bras suivi de la flexion du coude, par exemple. On voit que la trace finale dépend du degré de superposition des deux gestes. Il faut aussi remarquer que lorsque l'ordre des gestes est inversé, nous nous retrouvons avec une trajectoire illogique. En effet, nous pourrons observer une rotation horizontale autour du coude ce qui est impossible. Le fait d'inverser les deux gestes change la stratégie de génération et donc la position des cibles virtuelles. Le même phénomène est présent pour des mouvements en deux dimensions.

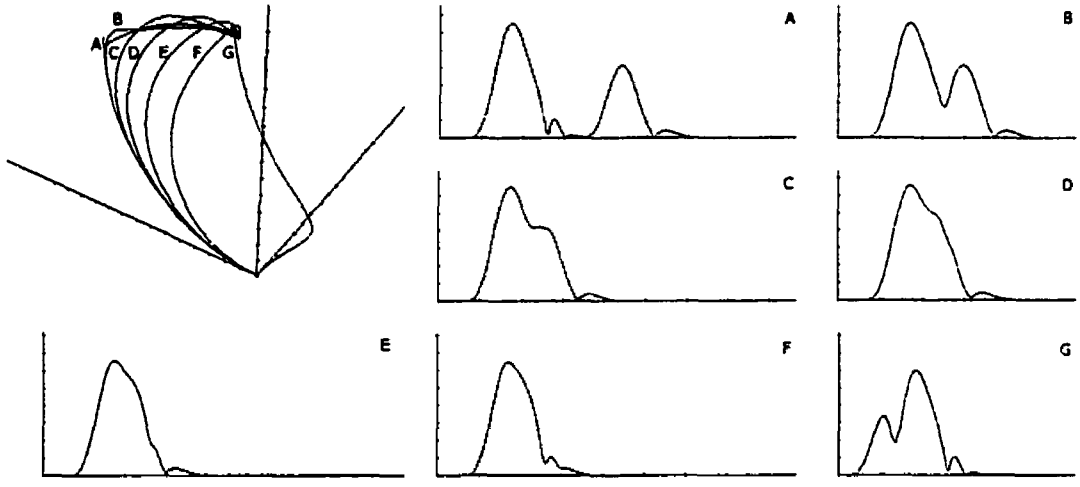


FIGURE 3.11 Simulations de mouvements complexes. Résultats de la superposition de deux mouvements simples. Premier mouvement : D_{11}=1,77; μ_{11}=-0,6; σ_{11}=0,2; t_{01}=-0,1; D_{21}=-0,2; μ_{21}=-0,25; σ_{21}=0,05; C_{01}=1; θ_{01}=0; ϕ_{01}=0; ρ_{01}=0. Deuxième mouvement: D_{12}=0,93; μ_{12}=-0,5; σ_{12}=0,15; D_{22}=-0,15; μ_{22}=-0,2; σ_{22}=0,1; C_{02}=2; θ_{02}=1,57; ϕ_{02}=1,57; ρ_{02}=0. A)t_{02}=0,50 B)t_{02}=0,15 C)t_{02}=0,05 D) t_{02}=0,00 E)t_{02}=-0,05 F)t_{02}=-0,10 G)t_{02}=-0,30

Nous voyons donc que, théoriquement, le modèle génère des trajectoires qui pourraient être observées pour des mouvements humains en trois dimensions. Les simulations nous permettent aussi de voir de quelle manière la trajectoire peut changer selon le délai entre l'activation de deux synergies différentes.

3.7 Conclusion

Dans ce chapitre, nous vous avons présenté le modèle delta-lognormal en trois dimensions. Dans un premier temps la généralisation de plusieurs modèles a été analysée et il en est ressorti que le modèle delta-lognormal était le plus apte à décrire les mouvements complexes en trois dimensions. Par la suite, deux modèles intermédiaires ont été proposés pour étudier les gestes en trois dimensions, mais n'ont pas été retenus. Ils étaient basés sur une définition mathématique du geste simple. Par contre, en définissant ce dernier comme étant un geste ne faisant intervenir qu'une seule synergie, donc deux systèmes neuro-musculaires, un agoniste et l'autre antagoniste, agissant dans un seul plan, nous avons élaboré le modèle final comportant onze paramètres. Une brève présentation d'un extracteur a été faite. Celle-ci a été suivie par des simulations de gestes en trois dimensions. Le chapitre suivant présente le protocole expérimental ainsi que des résultats obtenus sur des gestes simples réels.

Chapitre 4

Résultats expérimentaux

4.1 Introduction

Dans le chapitre précédent, nous avons proposé et simulé le modèle delta-lognormal en trois dimensions pour décrire les mouvements humains. Le présent chapitre est consacré aux résultats expérimentaux qui nous permettent de vérifier la validité du modèle proposé. Nous débuterons par la description du matériel utilisé ainsi que par celle des méthodes et des différents outils pour le traitement des résultats. Par la suite, les gestes choisis pour l'acquisition seront présentés. Le chapitre se terminera par les résultats obtenus et comprendra une comparaison entre les différents gestes simples en plus d'une critique du modèle et les conditions expérimentales utilisées.

4.2 Matériel utilisé

Avant de passer aux résultats obtenus, nous allons d'abord décrire le matériel ainsi que les logiciels utilisés. Dans cette section nous allons présenter le fonctionnement de l'appareil d'acquisition, le système "A Flock of Birds" de la compagnie Ascension. Celui-ci comporte cinq composantes principales : les capteurs, le générateur de champs magnétique, la boîte de contrôle du champ magnétique, la boîte d'acquisition et l'ordinateur pour visualiser les résultats.

4.2.1 Description des composantes de l'appareil d'acquisition

L'élément clé qui contrôle le système d'acquisition en trois dimensions est la boîte d'acquisition (MCU : Motion Control Unit). Elle est la boîte noire en dessous de l'écran sur la figure 4.1. La MCU contient une carte d'acquisition pour chaque capteur du système ainsi qu'une interface qui permet d'envoyer des commandes pour contrôler le champ magnétique généré. Ce système est en mesure de connaître la position des capteurs ainsi que leur orientation, car il connaît la force du champ magnétique généré et analyse les courants induits dans les capteurs. Le système contient aussi une carte réseau éthernet pour être en mesure de communiquer les valeurs acquises à une fréquence fixe à un ordinateur contenant un logiciel permettant de visualiser les signaux.

Le deuxième élément du système est la boîte de contrôle du champ magnétique. (ERC : Extended Range Controller). Elle est placée à la droite du MCU sur la figure 4.1. Le ERC est en fait un amplificateur. Il contient une entrée venant du MCU ce qui lui permet de recevoir les commandes à exécuter. Une sortie à haute tension est connectée sur le générateur de champs magnétiques. Il est possible de contrôler trois champs directionnels.

FIGURE 4.1 Système d'acquisition (MCU et ERC). La boîte noire sous l'écran est le MCU et la boîte grise à droite est le ERC.

La boîte illustrée sur la figure 4.2 est le générateur de champs magnétique (ERT : Extended Range Transmitter). Celle-ci contient des circuits permettant de générer des champs magnétiques directionnels qui sont contrôlés par la sortie haute tension de l'ERC. Ces champs sont produits par des impulsions de courant continu, car ils sont moins sensibles aux distorsions que ceux générés par du courant alternatif.

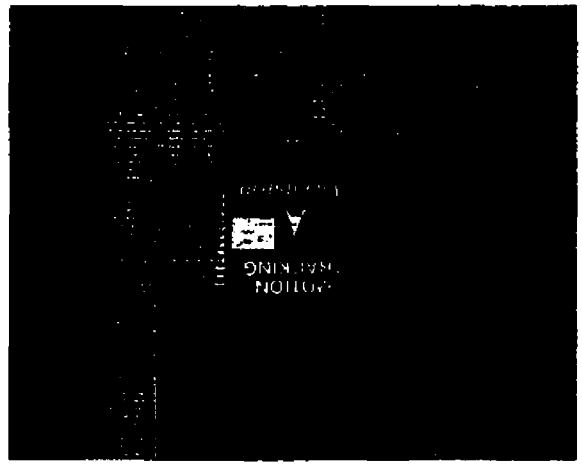


FIGURE 4.2 Système d'acquisition (ERT). Le ERT est la boîte qui est posée sur un support de forme cylindrique.

Enfin, les capteurs, que l'on peut voir sur la figure 4.3, produisent un courant qui varie selon leur position dans l'espace. Ces capteurs sont composés de trois bobines placés orthogonalement. Ces bobines produisent un courant qui aura une intensité et une direction qui dépendra du champ appliqué ainsi que la position et l'orientation de celles-ci.

4.2.2 Théorie du fonctionnement de l'appareil d'acquisition

Pour connaître la position et l'orientation des capteurs, la MCU utilise le principe de l'induction électromagnétique. Nous savons qu'une bobine soumise à un champ magnétique variable produit un courant qui est proportionnel à la puissance du champ et à l'orientation de la bobine par rapport à celui-ci. Une bobine placée parallèlement aux lignes

de champs produira un courant maximal alors qu'une en position perpendiculaire, elle ne produira aucun courant. Comme nous avons trois bobines en position orthogonale dans un capteur, nous pouvons comparer l'intensité et la direction des courants pour connaître la position et l'orientation de celui-ci. Cependant, pour un champ magnétique donné, nous pouvons placer un capteur à des endroits et des orientations différents et obtenir des courants identiques provenant des bobines. Pour résoudre ce problème, le système génère trois champs magnétiques directionnels successifs dans des directions orthogonales. Ceci permet d'éliminer des possibilités et d'en arriver à un seul couple de solutions, car la polarité du champ généré est inconnue. Une solution unique est trouvée en assumant que tous les points sont produits dans un seul hémisphère qui correspond aux solutions dont la position sur l'axe des X est positive. Il faut donc faire attention de ne pas dépasser la frontière lors des expérimentations.

4.2.3 Choix de la position des capteurs sur le sujet

Nous venons de voir de façon générale la manière dont fonctionne le système d'acquisition pour connaître la position des capteurs. Maintenant, voyons plus en détail les choix que nous avons faits pour le positionnement de ceux-ci sur le sujet. La figure 4.3 montre les capteurs qui ont été installés sur un sujet. Nous voyons qu'ils sont maintenus par des attaches velcro. D'autres attaches servent à retenir les fils pour ne pas que le sujet soit gêné dans ses mouvements. Malheureusement, il est difficile de maintenir les capteurs exactement en place, car le fait de bouger provoque un déplacement de ceux-ci. Il est évident que nous ne pouvions pas placer les capteurs sur les vêtements, car le déplacement des capteurs aurait été accentué.

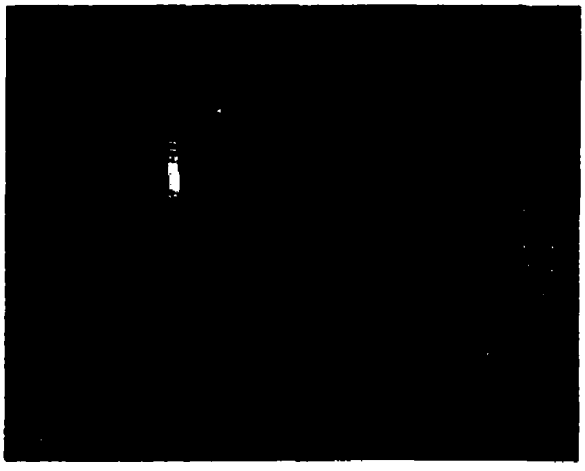


FIGURE 4.3 Positionnement des capteurs. Le capteur est retenu au poignet du sujet au moyen d'une attache Velcro.

Nous avions six capteurs à notre disposition, mais seulement trois ont été utilisés. Le premier a été placé à l'extrémité du membre produisant le mouvement. Cette position permet de maximiser le déplacement et augmente ainsi le rapport signal sur bruit. Les deux autres capteurs ont été placés sur le corps du sujet pour s'assurer que ce dernier ne ferait que le mouvement demandé. Ceux-ci devaient donc avoir une position fixe. Cependant, comme les capteurs se déplaçaient lors de l'exécution des mouvements, il a été impossible de différencier le déplacement du capteur de celui de la personne.

4.2.4 Discussion à propos de l'emplacement de l'appareil

Le système d'acquisition est, d'après le manufacturier, performant. Cependant, le fait qu'il utilise des champs magnétiques pour faire l'acquisition peut causer de sérieux problèmes. En effet, de nos jours beaucoup d'appareils génèrent eux-mêmes des champs qui peuvent causer des interférences avec l'appareil d'acquisition. De plus, toute structure de métal cause des distorsions. Il ne faut pas oublier l'électricité statique qui peut fausser

les résultats en induisant des courants dans les fils qui relient les capteurs au MCU. Il faudrait donc, idéalement, installer la machine dans une grande pièce et mettre toutes les sources d'interférence dans une autre. Malheureusement, un tel local n'était pas à notre disposition à l'École. L'appareil a donc été installé dans un local relativement petit. À cause des structures en métal et de la position de l'ordinateur, il a été nécessaire de faire très attention lors des acquisitions pour avoir des résultats les plus précis possibles. Le fabricant nous affirme que l'appareil est précis à environ un centimètre. Notre précision de lecture est moindre à cause de l'environnement dans lequel l'appareil a été installé. Nous n'avons pas été en mesure de calculer cette précision, car elle variait et dépendait de l'endroit où nous nous trouvions. Nous avons cependant pris soins de faire les tests dans la zone où le bruit et la distorsion étaient minimaux.

4.3 Traitement des résultats

Dans la section précédente, nous avons présenté l'appareil d'acquisition ainsi que son mode de fonctionnement. Comme l'appareil nous fourni des résultats sous une forme brute, il nous faut les traiter pour être en mesure de les analyser. Cette section traite des différents programmes utilisés pour le traitement des données recueillies. Nous présenterons toutes les étapes nécessaires pour arriver aux résultats présentés plus loin.

4.3.1 Acquisition des données

L'appareil d'acquisition fournissait la position et l'orientation des capteurs à une fréquence fixe choisie par l'utilisateur. Celles disponibles vont de 75 Hz à 144 Hz environ et sont des valeurs discrètes. Le fabricant nous propose d'utiliser une fréquence d'échantillonnage de 76,2 Hz ou de 86,1 Hz, car ce sont celles qui offrent la meilleure performance. Malgré ces recommandations, nous avons choisi de faire un échantillonnage à 97,9 Hz pour avoir une fréquence semblable à celle des tablettes à numériser qui ont été

utilisées lors de l'analyse de l'écriture. Les mêmes outils de traitement de signal pourront donc être utilisés en gardant à l'esprit qu'il y a une légère différence sur la fréquence d'échantillonnage.

À la fréquence choisie, la position et l'orientation de tous les capteurs sont calculés par le MCU et envoyés via une connexion réseau éthernet à un poste de travail. Le poste qui est à notre disposition est un ordinateur ayant un Processeur Intel Pentium II cadencé à 350 MHz. Il contient 64 Mo de mémoire vive, deux cartes réseaux éthernet 10 Mbits ainsi qu'un disque de 4 Go qui nous permet de stocker les données recueillies. Nous utilisons le logiciel 6D-Research (Skill Technologies) pour être en mesure de récupérer les points transmis par le MCU et de visualiser les mouvements exécutés en temps réel.

Le logiciel nous permet de visualiser les capteurs dans l'espace. Nous pouvons associer un os à chaque capteur pour avoir une meilleure idée des mouvements exécutés. Le logiciel nous permet de calculer plusieurs paramètres, par exemple, la vitesse et l'accélération linéaires. Il permet aussi de filtrer les données à l'aide d'un filtre numérique de Butterworth (Winter, 1979) ayant une fréquence de coupure de 6 Hz. Celui-ci ne nous a servi que pour une analyse visuelle préliminaire des données. Lors de cette analyse, lorsque nous soupçonnions un problème d'acquisition, il nous était possible de faire reprendre les gestes immédiatement.

La figure 4.4 montre l'interface du logiciel. Nous voyons, à gauche, les parties du squelette auxquelles nous avons associé des capteurs. Le point de vue peut être modifié à l'aide de déplacements de la souris par l'utilisation les boutons à droite. Il nous permet de voir en temps réel les capteurs, de faire l'acquisition de données ou de calculer les variables préétablies.

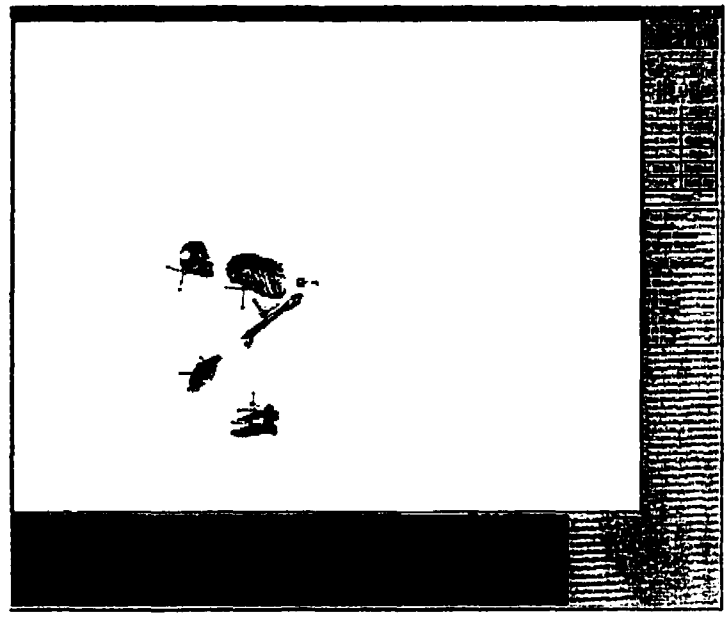


FIGURE 4.4 Interface du programme d'acquisition.

Le programme a été utilisé pour acquérir les données. Pour chaque acquisition, un certain nombre de fichiers sont générés. Celui qui nous intéresse contient les valeurs brutes de position et d'orientation des capteurs. Les autres contiennent tous les résultats des calculs demandés, par exemple, la vitesse, l'accélération ou le déplacement. Le nombre de fichiers ainsi que l'espace disque nécessaire pour le stockage augmente très rapidement. Pour faciliter la gestion des fichiers, les gestes ont été regroupés. Chaque fichier contient donc cinq répétitions de deux gestes simples, soit dix gestes simples. Le fichier contenant les données brutes a l'extension .RAW. Les positions du premier capteur se trouvent dans les colonnes 10, 11 et 12 pour la position en X, Y et Z respectivement. L'origine et l'orientation du système d'axe sont définis par la position et l'orientation de l'ERT.

4.3.2 Filtres numériques utilisés

Pour l'analyse des signaux en trois dimensions, nous avons utilisé les filtres que nous avions pour le traitement de l'écriture, car la fréquence d'échantillonnage des

mouvements (97,9 Hz) est proche de celle utilisée pour l'analyse de l'écriture (100 Hz). Par conséquent, le fait d'avoir une fréquence d'échantillonnage différente modifie la fréquence de coupure du filtre passe bas. Celle-ci passe de 16 Hz pour un signal échantillonné à 100 Hz à 15,66 Hz pour celui à 97,9 Hz. Cette modification est minime et n'influe pas sur les résultats, car les fréquences contenues dans le signal dépassent rarement 8 Hz.

Pour le filtre dérivatif, la modification de la fréquence d'échantillonnage produits des vitesses différentes, mais elles sont à un facteur d'échelle près des véritables valeurs. Cette erreur n'est pas importante dans le cas présent, car nous tentons de prouver que les courbes observées sont de forme delta-lognormales et non de trouver les paramètres exacts de celles-ci. L'approximation est donc acceptable dans ce contexte. Par contre, lors d'études subséquentes portant sur l'interprétation des paramètres, il faudra corriger les coefficients des filtres pour les ajuster à la fréquence d'échantillonnage utilisée.

4.3.3 Séparation des mouvements simples

Nous avons écrit un programme qui permet de séparer chaque mouvement simple qui a été acquis. Le programme extrait la position, en X, Y et Z, du capteur à partir du fichier de données brutes (extension .RAW) et affiche la vitesse, calculée en utilisant l'équation (4.1), qui correspond à la distance entre deux points divisée par le temps entre deux échantillons, à l'écran. Nous avons utilisé les librairies graphiques de Borland pour l'affichage. Le choix des positions de coupure du signal est fait manuellement, de manière visuelle. L'utilisateur place un curseur au début du signal correspondant à un mouvement et l'autre à la fin et il doit noter la position des deux curseurs affichés dans le coin supérieur gauche de l'écran, car ce sont ces valeurs qui permettront de séparer les signaux. Le code du programme est disponible à l'annexe 3.

$$V=97,9*\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2} \qquad (4.1)$$

La vitesse n'est pas filtrée, mais il est possible de bien voir les endroits où l'on doit séparer les mouvements. En effet, lors de l'exécution de ceux-ci, la personne fait un arrêt entre chaque geste, ce qui génère une vitesse presque nulle. La figure 4.5 montre un exemple de courbe que l'on doit séparer. On voit bien où commence et se termine chaque mouvement. Dans les cas où la séparation n'était pas aussi nette, cas assez rare, nous avons positionné la coupure au minimum de vitesse.

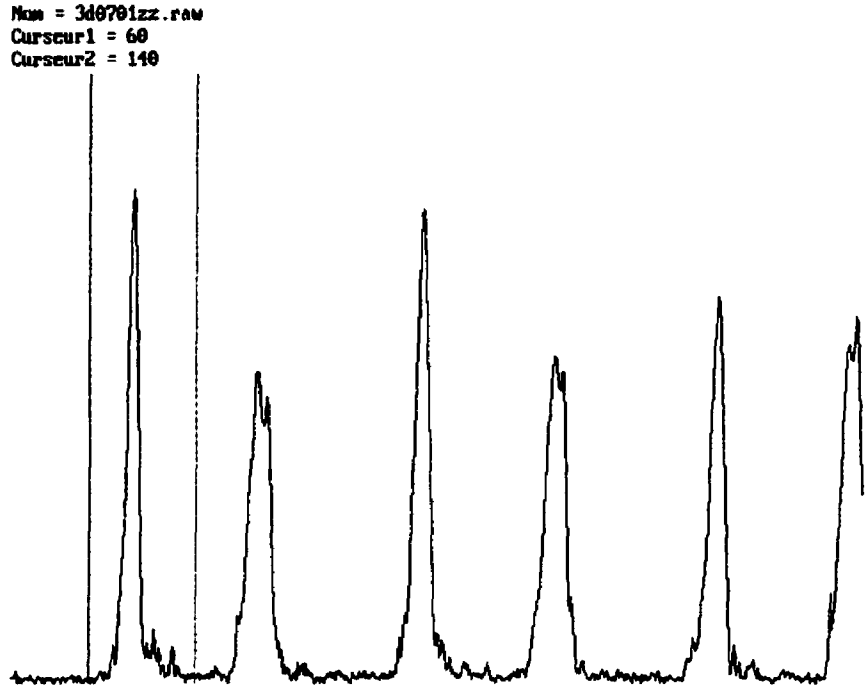


FIGURE 4.5 Séparation des gestes simples. Chaque geste simple peut être extrait facilement à cause des temps d'arrêt entre chacun.

Les positions choisies pour la séparation sont ensuite passées en paramètre à un script que nous avons conçu et dont le code est disponible à l'annexe 4. Ce script isole les points de chaque mouvement et crée un fichier distinct pour chacun. Le nom de tous les fichiers générés par ce script ont la même nomenclature : 3DGGPPEE.RAW. 3D signifie que ce fichier contient des coordonnées en trois dimensions qui sont placés en colonnes.

La première colonne contient les coordonnés en X, la seconde, en Y et la troisième, en Z. Chaque geste est numéroté et GG est le numéro de celui dont les positions successives sont contenues dans le fichier. Comme plusieurs personnes ont été testées, PP représente le numéro de la personne, tandis que EE est le numéro de la répétition du geste.

4.3.4 Extraction des paramètres des delta-lognormales

Après avoir séparé les gestes, nous avons fait une inspection visuelle des profils de vitesse pour distinguer les gestes ayant un seul pic de vitesse principal de ceux en ayant plusieurs. Comme un geste simple ne doit comporter qu'un seul maximum de vitesse principal ainsi qu'un maximum de deux maximums secondaires, tous les profils à pics principaux multiples ont été mis de côtés et n'ont pas été analysés. Nous avons ensuite extrait les paramètres du modèle delta-lognormal qui permettaient la meilleure reconstruction du profil pour les courbes n'ayant qu'un seul maximum de vitesse. Nous avons utilisé le programme écrit par Guerfali (1999) pour trouver les paramètres.

Le programme d'extraction n'est pas en mesure de lire des fichiers contenant des données sur des mouvements en trois dimensions. Par contre, il peut lire des fichiers d'un format spécifique qui contiennent directement le profil de vitesse. Il nous a donc été nécessaire d'écrire un programme qui fait la conversion et le code de celui-ci est disponible à l'annexe 5. En gros, le programme lit les points d'un fichier des données en trois dimensions (extension .RAW). Il calcule ensuite les dérivés de X, Y et Z par rapport au temps à l'aide du filtre dérivatif. Il calcule la vitesse à l'aide des dérivés trouvés et filtre le signal obtenu à l'aide du filtre passe bas. Il écrit enfin le fichier de données pour l'extracteur. Le fichier a l'extension .BRT et comporte 5 colonnes. La première correspond à la coordonnée de temps et chaque échantillon est à une distance de 0,01 s de la précédente. La seconde, troisième et quatrième colonnes ne sont pas utilisés, mais contiennent les

valeurs 5, 1 et 1 respectivement pour maintenir la compatibilité. La dernière colonne contient les valeurs du profil de vitesse calculé.

Le programme d'extraction, en plus de faire l'extraction des paramètres, permet de visualiser la courbe de vitesse originale et celle reconstruite à l'aide du modèle delta-lognormal. Il utilise la méthode d'estimation graphique de Wise (1966) suivie de l'optimisation non-linéaire pour extraire les valeurs optimales des paramètres D_1, μ_1, σ_1, t_0, D_2, μ_2 et σ_2. En plus des paramètres, l'erreur quadratique moyenne est calculée. La figure 4.6 montre l'interface du programme.

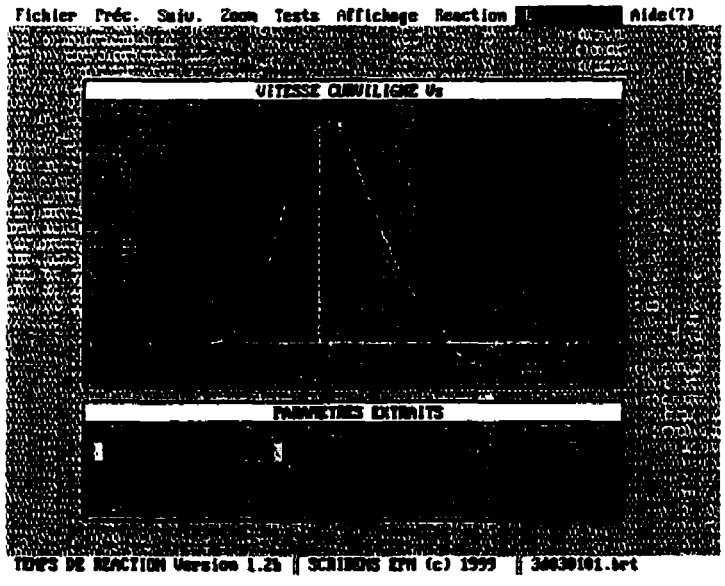


FIGURE 4.6 Interface du programme d'extraction delta-lognormal.

La méthode d'estimation graphique de Wise (1966) consiste à utiliser les pentes aux points d'inflexion sur la partie montante et descendante du profil de vitesse. Il est possible d'estimer l'aire sous la courbe delta-lognormale en calculant l'aire du triangle formé par les deux droites tangentes aux points d'inflexion et l'axe des X et en la multipliant par un facteur de proportionnalité qui dépend du paramètre σ_1 . Ce dernier est évalué à l'aide

d'une table et dépend des pentes. Ensuite, il est possible d'estimer μ_1 à l'aide d'une formule faisant intervenir les pentes et σ_1. Le calcul du paramètre t_0 fait, pour sa part, intervenir un paramètre qui vient d'une table ainsi que les pentes aux points d'inflexion. La méthode est donc assez simple et directe pourvu que nous ayons les tables utilisées par Guerfali et Plamondon (1993) à notre disposition.

L'optimisation non linéaire utilise un principe assez simple. Il consiste à modifier les paramètres connus de sorte que l'erreur globale diminue. Pour être en mesure de savoir quelle valeur ajouter ou retrancher à un certain paramètre, on utilise les dérivées partielles. Plus la dérivée partielle donne une valeur importante, en valeur absolue, plus la correction sera petite, car nous sommes dans une zone où l'erreur change rapidement. Il est donc possible, après plusieurs itérations, de trouver des paramètres qui nous donnent un minimum local pour l'erreur de reconstruction. Cependant, comme l'espace est discontinue, la solution trouvée dépendra en grande partie des paramètres initiaux.

Le logiciel d'extraction contient aussi une routine qui est intéressante pour certains gestes. Celle-ci permet de déterminer, pour un profil de vitesse strictement positif, dû à l'utilisation du module, les parties qui auraient dû être des vitesses négatives avant de faire l'extraction. Cette supposition est logique pour plusieurs mouvements pour lesquels nous observons des oscillations à la fin, mais pour d'autres nous obtenons des paramètres erronés à cause de celle-ci. Il faut donc faire attention lors de l'analyse des résultats.

Dans la majorité des cas, les courbes reconstruites, à l'aide du modèle delta-lognormal, en utilisant les paramètres extraits, correspondent bien aux courbes originales. Cependant, dans certains cas, aucun paramètre ne peut être extrait, car l'estimation graphique échoue. Ceci est arrivé pour quelques gestes et nous n'avons pas tenu compte de ceux-ci lors des analyses.

4.3.5 Programme de visualisation en trois dimensions

Enfin, nous avons écrit un programme qui nous permet de visualiser les traces des gestes en trois dimensions et d'extraire manuellement les paramètres permettant la reconstruction tri-dimensionnelle du tracé observé. Le code du programme est disponible à l'annexe 7. Nous avons utilisé la programmation WIn32 pour la création et la gestion des fenêtres et des boutons de l'interface. La fenêtre permettant la visualisation en trois dimensions des tracés a été implantée en utilisant les librairies OpenGL. L'interface est illustrée à la figure 4.7. À gauche, il y a les cases à cocher pour nous permettre de choisir ce que nous désirons visualiser. Les boutons permettront de contrôler les étapes de calcul pour l'extraction automatique qui sera implantée éventuellement. Au centre, la fenêtre du haut permet d'afficher les traces en trois dimensions à l'aide des routines OpenGL et celle du bas, les profils de vitesse. Les paramètres qui peuvent être modifiés ainsi que d'autres informations sur le fichier courant sont affichées à droite.

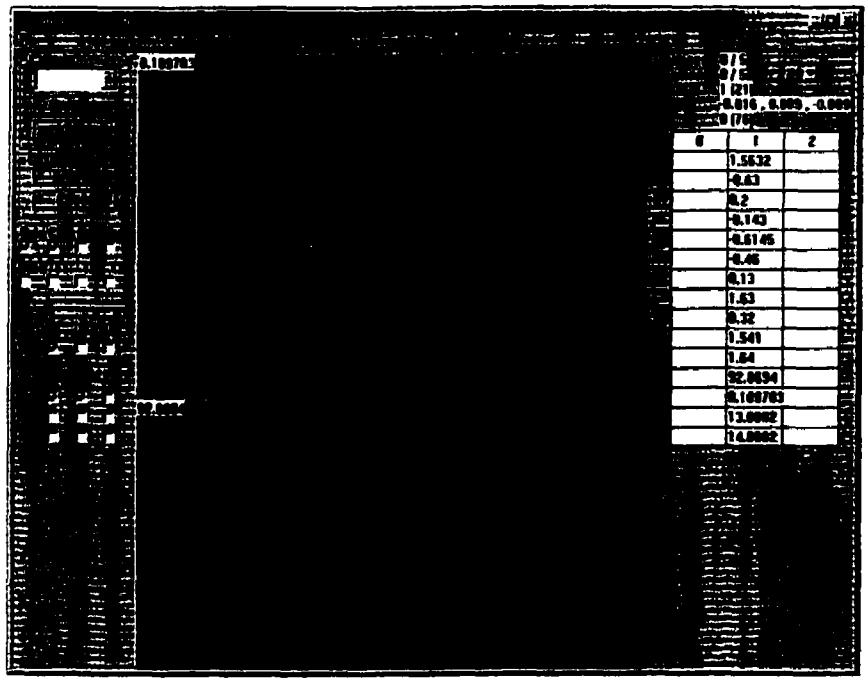


FIGURE 4.7 Interface du programme de visualisation 3D.

Le programme ne fait aucune extraction automatique de paramètres pour l'instant. Cependant, il a été conçu dans le but d'inclure une telle fonctionnalité et pourra être utilisé dans d'autres recherches. Le programme permet de modifier les paramètres de chaque geste simple et de voir l'influence immédiate sur la reconstruction. Il est possible de changer le point de vue autant pour le profil de vitesse que pour la trace en trois dimensions. De plus, il est possible de faire un zoom sur les parties du tracé qui nous intéressent. Le programme peut charger un fichier contenant des points en trois dimensions et, si tel est le cas, il calcule l'erreur quadratique moyenne sur le tracé ainsi que sur le profil de vitesse en comparant les données reconstruites et celles issues du fichier.

Nous avons donc présenté le matériel ainsi que les logiciels utilisés pour l'acquisition des données et le traitement des résultats. Nous voyons que la procédure pour l'extraction des paramètres est complexe, car plusieurs logiciels entrent en jeu. Un programme intégrant toutes les opérations d'acquisition et d'extraction aurait été intéressant, mais comme le code source nécessaire pour nous permettre de réaliser ce projet n'était pas disponible, il nous a été impossible d'unifier le tout.

4.4 Protocole expérimental

Maintenant que nous avons présenté le matériel et les logiciels utilisés pour l'analyse des résultats, nous allons décrire le protocole expérimental utilisé. Celui-ci doit être robuste et fiable et doit permettre de s'assurer que chaque expérience sera exécutée dans les mêmes conditions ou des conditions semblables aux autres. Au départ, dans une phase exploratoire où nous voulions nous familiariser avec les appareils, nous n'avions pas planifié de protocole expérimental. Les acquisitions se sont donc fait sur des gestes disparates. Il était, par conséquent, difficile de comparer les personnes, les gestes ou les membres. Dans une seconde phase, nous avons structuré notre approche et nous avons préparé une liste de gestes qui devaient être exécutés par chaque sujet.

Comme nous voulions analyser les mouvements simples, il a été nécessaire de trouver ceux qui ne faisaient intervenir qu'une seule synergie comme le stipule notre définition. Il est cependant difficile de savoir si plus d'une synergie a été sollicitée. Nous avons choisi de faire exécuter des gestes très simples aux sujets pour réduire au minimum les chances qu'une telle possibilité se présente et que nous ayons affaire à des gestes complexes.

La figure 4.8 montre l'ensemble des gestes choisis. Les photos sont regroupées deux par deux et la figure comporte deux colonnes. La photo de gauche pour chaque groupe montre la position initiale du sujet tandis que celle de droite, celle finale. Le mouvement inverse a aussi été acquis et analysé. De plus, les personnes ont exécuté les mouvements avec les membres du côté droit du corps ainsi qu'avec ceux du côté gauche.

Pour chaque acquisition nous demandions à la personne de s'installer confortablement dans la zone d'acquisition où le bruit de lecture et la distorsion étaient minimaux. Par la suite, nous lui demandions de répéter le mouvement jusqu'à ce qu'elle soit en mesure de l'exécuter confortablement. Aucune structure n'a été utilisée pour maintenir le corps stable ou pour immobiliser les articulations. La personne devait donc, en plus de se concentrer pour exécuter le geste le plus rapidement possible, tenter de rester immobile. Cette contrainte augmentait la complexité d'exécution de certains gestes.

Nous demandions au sujet de faire cinq répétitions successives du geste qui comportait un aller et un retour à la position initiale. Chaque déplacement entre deux positions devait se faire le plus rapidement possible en essayant d'aller pointer des cibles virtuelles dans l'espace. La personne devait maintenir chaque position pour environ une demi seconde de manière à ce que le membre se stabilise et qu'il soit facile de séparer les gestes simples lors de l'analyse. Cet arrêt pouvait provoquer, entre autre, une perte

d'équilibre lorsque nous testions les mouvements des jambes ce qui entraîne un temps d'arrêt plus petit dans ces conditions.

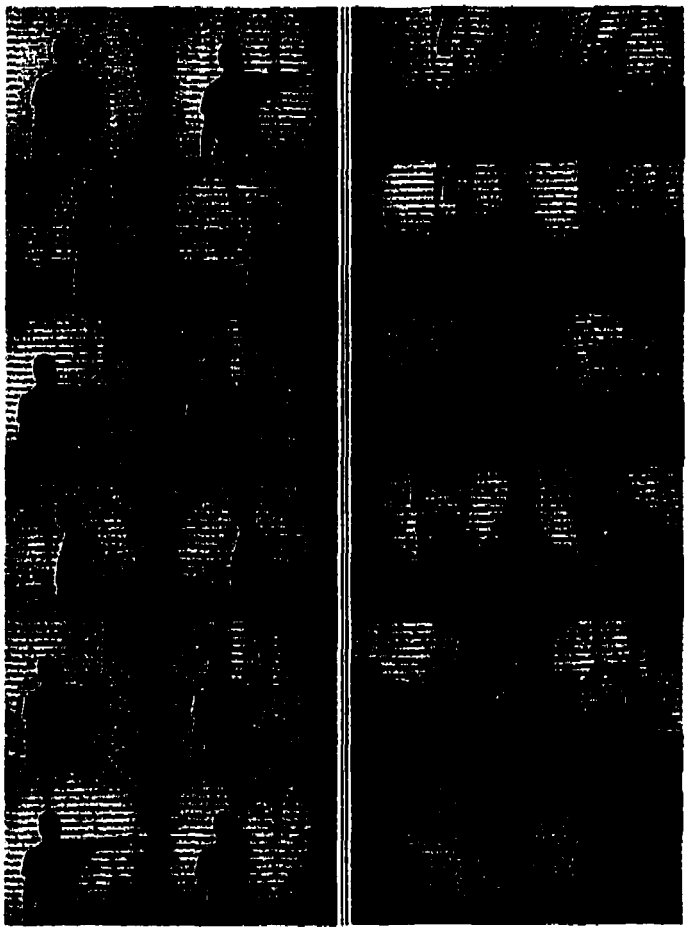


FIGURE 4.8 Gestes utilisés pour l'expérimentation. Les numéros au-dessus ou en dessous des flèches représentent le numéro du geste tel qu'utilisé pour nommer les fichiers. On voit la position de départ et finale pour chaque geste.

Les séances d'acquisition complètes, pour une personne, en suivant le protocole expérimental, duraient environ quatre heures. Les séances étant relativement longues, les sujets démontraient de la fatigue vers la fin de celles-ci. Dans ce cas, nous observons des

mouvements moins rapides et moins précis. Pour résoudre le problème, nous avons réalisé les tests sur plusieurs séances d'une durée plus courte à la convenance des sujets. Cependant, le fait d'étaler les tests sur plusieurs séances a un effet sur les résultats. Nous soupçonnons que, dépendant de l'état physique et psychologique du sujet, les résultats peuvent différer sur une personne pour un même test. Cependant, cette hypothèse n'a pas été vérifiée dans cette étude préliminaire.

Les conditions expérimentales n'étaient pas optimales, mais pour ce que nous tentons de prouver, elles étaient suffisantes. Il va de soi que, si nous voulons comparer les performances des différentes personnes, il faudra élaborer un protocole expérimental beaucoup plus strict et se doter d'équipement qui permet un meilleur contrôle sur le sujet pour éviter que celui-ci ne fasse de mouvements involontaires. La suite du chapitre rapporte les résultats expérimentaux et nous présentons une analyse et une discussion.

4.5 Résultats pour des mouvements simples

Dans la section précédente, nous avons décrit le protocole expérimental utilisé pour faire l'acquisition des gestes qui seront analysés. Nous présentons, maintenant, les résultats de la reconstruction de mouvements simples en trois dimensions. Un échantillon de courbes caractéristiques est présenté dans les pages suivantes. En annexe se trouve un exemplaire typique de la courbe de vitesse de chaque geste échantillonné. De plus, pour les courbes de vitesse choisies, la reconstruction en trois dimensions est aussi présentée.

4.5.1 Reconstructions réussies

La figure 4.9 montre six reconstructions en trois dimensions de gestes simples. La courbe reconstruite est tracée en trait foncé, tandis que l'originale est plus pâle. Il en va de même pour les courbes de vitesse en dessous de la reconstruction. Ces dernières sont

strictement positives, car le calcul du module de vitesse ne nous permet pas de savoir quelles parties du profil de vitesse devraient être positives ou négatives. Il est à noter que le code en haut à gauche de chaque courbe est le nom du fichier dont la courbe a été extraite et suit la nomenclature décrite précédemment.

Pour plusieurs courbes, nous avons obtenu d'excellents résultats. Un bon exemple est le fichier 3D020305. On voit que la courbe reconstruite et celle originale se superposent aussi bien pour la trace en trois dimensions que pour le profil de vitesse. L'erreur quadratique moyenne pour la reconstruction de la trajectoire est aussi très petite (0,0070 cm^2). La courbe 3D030102 montre aussi un excellent résultat, mais si nous regardons l'erreur quadratique moyenne (0,1049 cm^2), celle-ci est environ quinze fois plus grande que la précédente. Plusieurs facteurs influencent l'erreur. Dans ce cas-ci, un décalage dans la position des points, qui n'est pas visible sur la figure, est probable, car nous avons du bruit avant la courbe delta-lognormale principale. C'est ce bruit qui déplace la courbe originale et qui peut provoquer un décalage. De plus, la trajectoire reconstruite est plus longue que l'originale. Ce phénomène apparaît lorsqu'il y a des artefacts en fin de signal. Nous enregistrons une oscillation qui augmente la distance totale, ce qui se reflète dans la courbe reconstruite sous la forme d'un dépassement. Plus la distance impliquée dans l'oscillation est importante, plus l'erreur est augmentée. Il faut donc faire attention et ne pas seulement se fier à l'erreur quadratique moyenne, car elle peut nous induire en erreur dans certains cas. Une inspection visuelle de la reconstruction est donc nécessaire.

Le fichier 3D230304 nous montre aussi un autre bon exemple des oscillations que nous pouvons observer à la fin d'un geste. On voit que le geste correspond au modèle, sauf pour le point final. Il est difficile de maintenir une position stable du bras pointant vers l'avant à cause de la gravité ce qui provoque des oscillations pour corriger et maintenir la main à une position fixe.

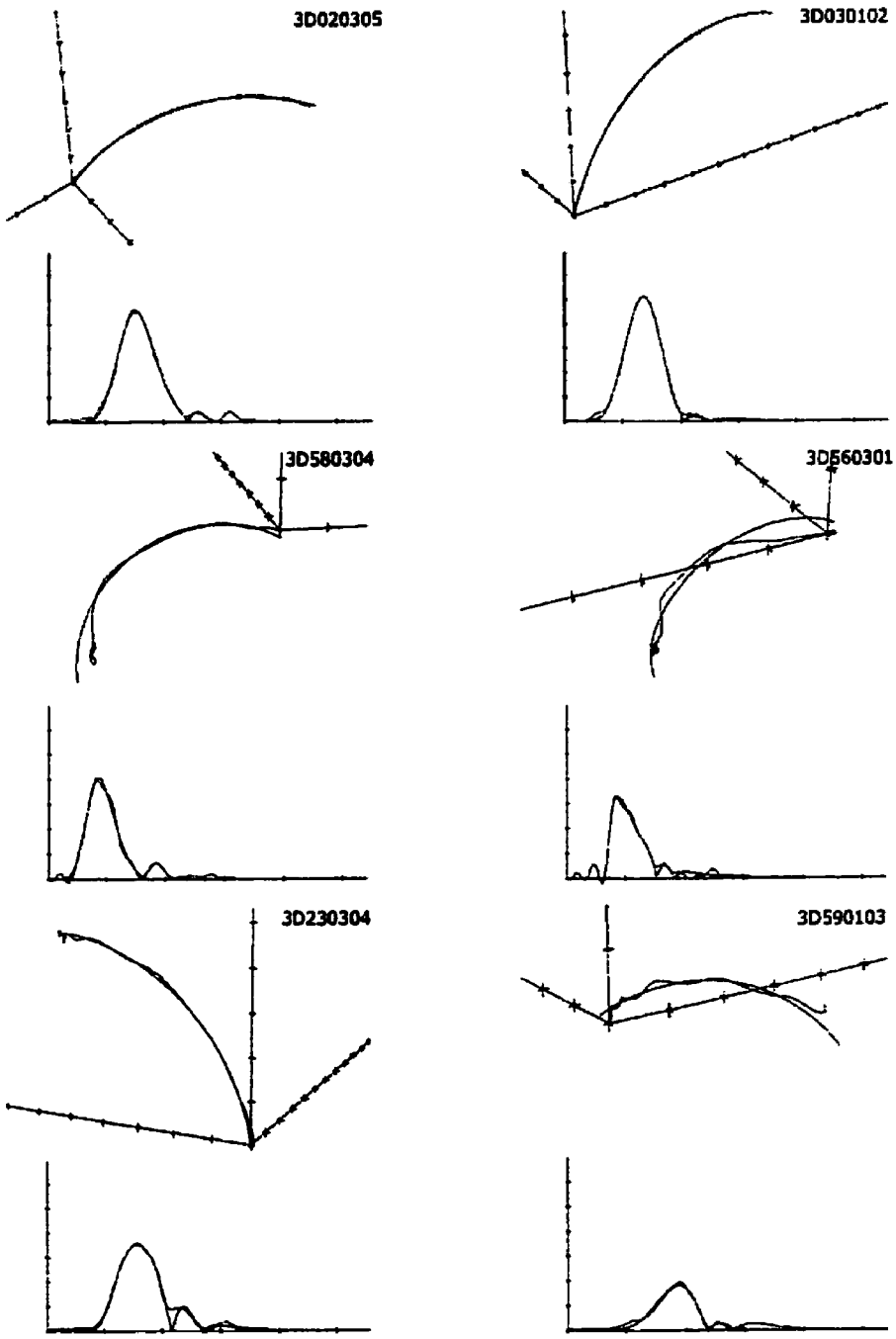


FIGURE 4.9 Exemples de reconstruction en 3D. Six mouvements reconstruits à l'aide du modèle. Les paramètres associés sont présentés dans l'annexe 2.

4.5.2 Cas problématiques

Dans plusieurs cas, nous observons aussi des oscillations durant le mouvement. Celles-ci peuvent être de différentes amplitudes et le fichier 3D590103 montre un cas extrême. Malgré cela, la reconstruction correspond assez bien au geste original. Dans le cas présenté, le geste consistait à lever le bras à partir d'une position où le bras et l'avant-bras étaient à l'horizontal. Ce geste est peu commun et la position est légèrement inconfortable ce qui le rend difficile à réaliser. Parmi les autres causes provoquant des oscillations, on peut noter la perte d'équilibre que provoque certains gestes et le fait de combattre la gravité pour maintenir une position.

Dans le fichier 3D580304 nous remarquons deux phénomènes. Le premier est le fait qu'il y ait des oscillations à la fin du geste et dont nous avons déjà parlé. Le changement d'orientation du geste un peu avant la fin constitue le second phénomène. Cette déviation soudaine nous laisse croire qu'il pourrait y avoir plus d'un geste simple dans le mouvement. Ceci semble être confirmé lorsque nous regardons le profil de vitesse. En effet, à la suite du pic principal de vitesse, un second pic est présent. D'après la position du second pic et le résultat de la reconstruction de la vitesse, il semble qu'il n'y ait qu'une seule courbe delta-lognormale. En effet, il est possible d'expliquer certains pics secondaires à l'aide du modèle. Pour des exemples, se référer à la figure 2.4. Dans le cas qui nous intéresse, il semble que ce soit deux delta-lognormales distinctes, car il y a changement de courbure et d'orientation dans la trace en trois dimensions.

Le dernier cas est un exemple où la reconstruction a été un échec. La courbe de vitesse du fichier 3D560301 semble avoir été reconstruite correctement à première vue. Les oscillations présentent au début du signal sont causés par un effet de filtre. Vers la fin, on observe aussi du bruit qui est provoqué par les oscillations autour de la position finale du geste. Cependant la reconstruction en trois dimensions est beaucoup plus mauvaise qu'à

quoi on pourrait s'attendre. Il y a donc deux possibilités, car il ne s'agit pas d'un geste simple. La première est qu'il s'agit d'un mouvement complexe et l'autre est que le geste ne corresponde tout simplement pas au modèle. À cause des oscillations présentes dans le mouvement il est difficile de juger, mais il semble que la première hypothèse soit plus probable. En effet, si nous regardons le profil de vitesse attentivement, nous pouvons voir, dans la phase de décélération, plusieurs plateaux. Ceci laisse présager qu'il y a superposition de plusieurs mouvements simples. Il semble y avoir trois gestes superposés si nous nous fions au profil de vitesse et ça semble être confirmé par la trace en trois dimensions. Cette dernière comporte une partie horizontale, suivie d'une en diagonale pour finir avec une partie presque verticale. On a donc trois courbures et directions différentes et probablement trois gestes simples superposés.

4.6 Comparaison des profils de vitesse

Nous venons donc de voir des exemples de reconstruction de gestes simples en trois dimensions. Notre approche semble bien fonctionner sauf pour quelques cas. Cependant, plusieurs gestes ont des courbes de vitesse ne correspondant pas au modèle. Dans la présente section, nous allons comparer différentes personnes, différents gestes et différents essais pour tenter de voir ce qui peut influencer la correspondance des courbes au modèle. Comme nous avons des résultats que pour deux sujets, ces résultats préliminaires ne nous permettrons pas de tirer de conclusion définitive, mais ils nous permettront d'avoir une bonne idée des pistes à suivre pour continuer ces travaux.

4.6.1 Résumé des observations sur les profils de vitesse

La table 4.1 résume les observations faites sur les courbes de vitesse. La table est en deux parties, soit une pour chaque sujet (01 et 03). Le numéro de la répétition du geste est inscrit en dessous de celui du sujet. Chaque rangée correspond à un geste et pour chaque

sujet le geste a été exécuté par le côté gauche du corps (colonne de gauche) et par le côté droit (colonne de droite). Pour le côté droit, il faut ajouter 50 au numéro du geste pour trouver le fichier correspondant. De plus les rangées ayant un fond gris correspondent aux mouvements des membres inférieurs.

Le code que nous avons utilisé est "O" pour les profils de vitesse qui correspondent très bien au modèle. Seuls les cas qui permettaient de dire sans équivoque qu'il s'agissait d'une courbe delta-lognormale étaient étiquetés de telle sorte. À l'autre extrême, nous marquions d'un "X" toutes les courbes ayant plusieurs pics de vitesse ou ne correspondant pas au modèle du geste simple. Les cas entre ces deux extrêmes étaient marqués d'un "-". Pour les cas qui sont marqués par "nd", nous avons eu des problèmes lors de l'acquisition et les profils de vitesse pour ceux-ci ne sont pas disponibles. Cette classification a été faite visuellement et ne s'appuie sur aucune valeur numérique. Il faut dire qu'il s'agissait ici d'une classification grossière qui permettait d'avoir une vue d'ensemble des données recueillies pour choisir les données sur lesquelles nous voulions faire des analyses plus poussées, comme les reconstructions de la section précédente. Cette table nous permet quand même de faire des observations intéressantes.

Pour les figures qui vont être présentées, seul le profil de vitesse original est montré. Comme il s'agit de comparer des courbes, nous avons cru bon, pour ne pas surcharger les graphiques, de ne garder que le profil original. De plus, les courbes qui permettaient une bonne reconstruction sont présentées en annexe avec les tracés en trois dimensions qui leur sont associés.

TABLEAU 4.1 Résultats de l'inspection visuelle des profils de vitesse

	Sujet 01										Sujet 03									
	Gauche					Droit					Gauche					Droit				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1	nd	nd	nd	nd	nd	O	-	O	O	O	O	-	O	-	O	O	-	-	O	O
2	nd	nd	nd	nd	nd	-	O	-	-	X	O	-	-	-	O	O	-	O	-	O
3	O	O	O	-	O	O	O	O	-	X	O	O	O	O	-	O	O	-	O	O
4	O	-	O	-	O	O	O	O	-	-	-	-	O	O	X	-	O	-	O	-
5	nd	nd	nd	nd	nd	-	X	-	X	X	X	O	O	-	O	X	-	-	X	-
6	nd	nd	nd	nd	nd	X	X	X	X	X	O	O	O	O	-	O	X	-	X	-
7	-	O	O	-	O	O	-	O	O	-	O	O	O	O	X	-	O	O	O	O
8	X	-	-	X	X	X	X	-	-	X	-	O	O	-	O	X	X	X	O	-
9	nd	nd	nd	nd	nd	-	-	O	X	-	O	-	X	X	O	X	X	-	X	X
10	nd	nd	nd	nd	nd	X	X	X	X	X	-	O	-	-	O	X	X	X	X	X
19	O	O	-	O	-	X	X	X	-	X	X	X	X	X	X	X	X	X	X	X
20	-	-	-	-	O	O	X	X	X	-	X	X	X	X	X	-	X	-	X	X
23	nd	nd	nd	nd	nd	X	X	X	X	X	-	O	O	O	O	O	-	-	O	O
24	nd	nd	nd	nd	nd	-	-	O	-	X	X	O	-	-	X	-	O	-	O	-

4.6.2 Comparaisons des profils de vitesse

La première chose que nous observons est que d'un essai à l'autre une personne peut avoir des résultats très différents. La figure 4.10 montre un exemple de ce phénomène. À gauche, nous voyons une courbe qui correspond bien au modèle, soit ayant une forme de cloche avec un pic secondaire. Lorsque nous avons fait la reconstruction, les résultats se sont avérés très bons. Par contre la courbe de droite a une forme plutôt triangulaire et le modèle permet une reconstruction médiocre. Ces différences sont présentes dans tous les gestes à différents niveaux. Parmi les causes de celles-ci, on note la fatigue et le manque de concentration du sujet.

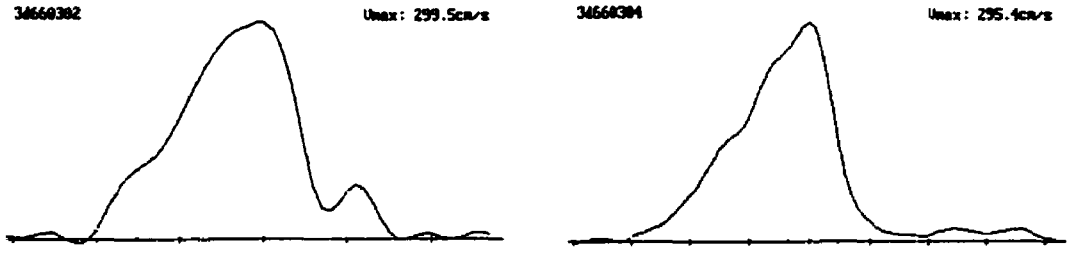


FIGURE 4.10 Variabilité entre les différents essais. Une personne exécutant deux fois le même mouvement ne génère pas des profils de vitesse identiques.

Habituellement, nous pouvons aussi nous attendre à ce que l'exécution des gestes soit plus facile d'un côté que de l'autre, car pour un droitier, par exemple, il aura tendance à utiliser plus souvent son bras droit. La table 4.1 nous montre qu'il y a effectivement une différence par rapport à la conformité des courbes au modèle. La figure 4.11 montre un cas particulier assez flagrant. La courbe de gauche correspond au modèle delta-lognormal tandis que celle de droite semble être une superposition de courbes. En effet, nous remarquons un palier lors de l'accélération ce qui nous permet de croire en la présence

d'une seconde courbe delta-lognormale. Nous obtenons d'ailleurs de très mauvais résultats lors de la reconstruction à l'aide du modèle pour un mouvement simple.

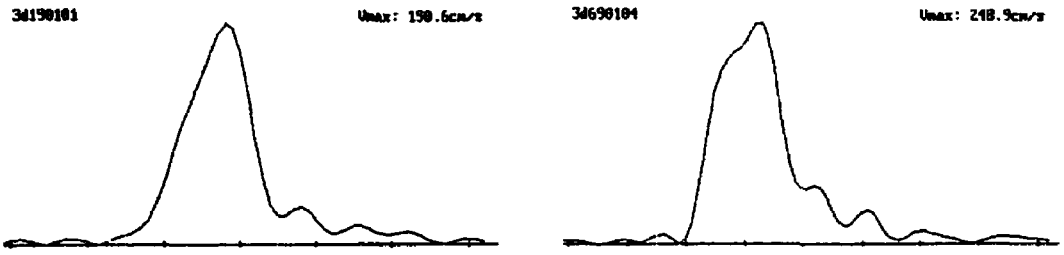


FIGURE 4.11 Comparaisons des membres de gauche et de droite. Les deux profils représentent le même mouvement. Celui de gauche a été exécuté par le poignet gauche et celui de droite par le droit.

Pour le sujet 01 nous observons aussi que le mouvement du poignet gauche donne de meilleurs résultats que celui du poignet droit ce qui semble inattendu sachant que la personne est droitière. En effet, nous nous attendons qu'un mouvement qui a été exécuté des milliers de fois tend de plus en plus à correspondre au modèle. Nous croyons que cette performance médiocre peut être causée par la fatigue du sujet, car le mouvement du poignet droit était l'un des derniers exécutés lors d'une séance d'acquisition.

À première vue, nous pourrions penser qu'un mouvement, à l'aller et au retour devrait générer des résultats semblables, car ce sont les mêmes groupes musculaires et la même articulation qui travaille. Dans plusieurs cas cependant, ce n'est pas ce qui est observé. En fait, si nous prenons le lever du bras comme exemple, il est vrai que son axe de rotation sera l'épaule dans les deux cas. Cependant, les groupes musculaires sollicités ainsi que la commande envoyée sera différente dans les deux cas. Cette commande tient compte du fait qu'il faut combattre la force gravitationnelle lors du lever et qu'il faut

ralentir lors de la descente. Ces commandes différentes font donc apparaître des profils de vitesse différents.

La figure 4.12 montre un exemple de courbe observée lors d'un aller-retour pour une rotation horizontale du bras autour de l'épaule. Nous voyons que pour l'aller nous n'avons pas une courbe delta-lognormale tandis qu'au retour la courbe correspond au modèle. Dans ce cas la gravité n'a pas d'effet, mais le fait que les groupes musculaires soient différents, tout dépendant de la direction, peut expliquer cette différence. Certains groupes sont plus forts que d'autres ce qui cause des différences sur la vitesse observée.

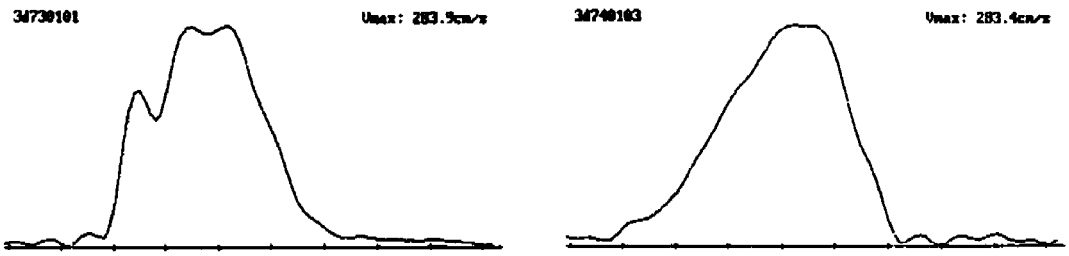


FIGURE 4.12 Influence de la direction du mouvement. Deux mouvements semblables, exécutés par une même personne, dans des directions différentes, génère des profils de vitesse différents.

Nous avons aussi tenté de comparer la performance des membres supérieurs aux membres inférieurs en rapport à la conformité au modèle. On voit, dans la table 4.1, que dans les cases au fond gris il y a beaucoup de courbes de vitesse qui ne correspondent pas au modèle. Par contre, dans les autres cases, nous remarquons le contraire. Comme les cases à fond gris représentent les mouvements des membres inférieurs, nous pouvons penser que le modèle ne s'appliquerait peut-être pas. Cependant, pour certains gestes, les courbes observées sont conformes. Il faut donc faire des études plus approfondies pour voir si le modèle s'applique directement ou non.

Il est quand même intéressant de voir une démarcation assez nette. Nous croyons que ceci est causé par l'utilisation que l'humain fait de chaque membre. Comme les bras sont utilisés habituellement pour des taches de précision, le contrôle de ceux-ci est plus précis. Les jambes sont utilisées pour la marche, donc elles ont besoins d'être fortes, pour soutenir le poids du corps, sans avoir à être aussi précises qu'une main qui nous permet, par exemple, d'écrire. Donc lorsque l'on demande à une personne d'atteindre une cible virtuelle, la précision est plus grande et les corrections de trajectoires sont moindres lors du déplacement d'un bras par rapport à celui de la jambe.

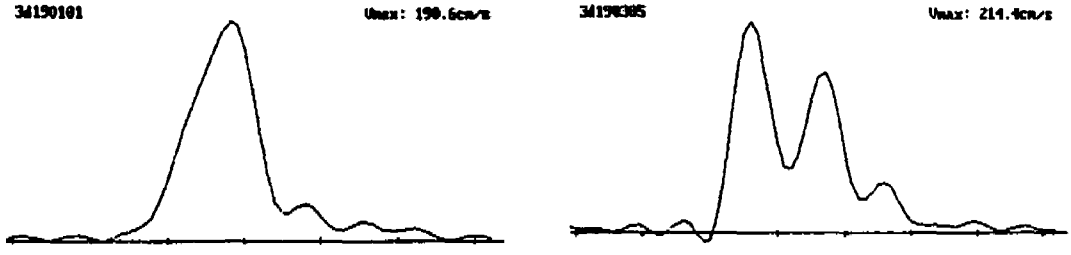


FIGURE 4.13 Comparaison des sujets. Pour un même mouvement, des sujets obtiennent des profils de vitesse très différents.

La dernière comparaison qui nous intéressait était celle entre les personnes. En fait nous voulions savoir si notre modèle pourrait éventuellement être utilisé pour différencier deux personnes. Même si les résultats sont très préliminaires, on peut déjà voir que pour un geste spécifique la courbe de vitesse d'une personne correspond au modèle alors que celle d'une autre ne peut être expliquée par le même modèle. La figure 4.13 nous montre un exemple de profil de vitesse observé chez deux personnes différentes pour un même geste. On voit que pour le sujet 1 le modèle delta-lognormal pour un geste simple peut permettre de reconstruire la courbe observée. Cependant, pour le sujet 3, on observe plusieurs maximums de vitesse ce qui laisse présager un mouvement complexe. Pour une même demande, des personnes différentes exécutent certains gestes différemment.

Pour les cas flagrants comme celui montré à la figure 4.13, il est facile d'affirmer que le modèle permettra de différencier les personnes. Cependant, des cas semblables sont peu nombreux comme nous pouvons le constater dans la table 4.1. Les résultats que nous avons recueillis ne nous permettent pas, pour le moment, de porter un jugement à savoir si le modèle permettra d'identifier des personnes ou s'il s'agit simplement de quelques cas d'exception.

4.6.3 Analyse de la courbure

Une chose intéressante, mais que nous avions prévu, est qu'il est possible de différencier certains membres à l'aide de la courbure. En effet, on sait que la courbure correspond à l'inverse du rayon du cercle tangent à la courbe. Pour un mouvement simple, le capteur a une position fixe sur le membre. La courbure que nous mesurons dépend de la distance entre le capteur et l'articulation autour de laquelle il tourne. Il est donc possible, en s'assurant de toujours positionner les capteurs aux mêmes endroits, de différencier les membres d'une personne en ne regardant que la courbure. On voit, dans la table des paramètres extraits présentée en annexe, que la courbure passe du simple au double dépendant si on fait un mouvement autour de l'épaule ou du coude. On remarque aussi que la courbure est très élevée pour un mouvement du poignet, car le capteur est près de l'articulation.

Cette observation est peut être utilisée pour différencier le mouvement d'un bras de celui d'une jambe, mais on ne pourra pas savoir si c'est un membre du côté gauche ou du côté droit qui a bougé. Même si de légères différences existent entre les deux côtés, les erreurs de positionnement des capteurs et d'acquisition des données font disparaître cette différence. Il faudra donc utiliser d'autres paramètres pour caractériser les membres, mais la courbure semble un bon point de départ.

4.7 Discussion

Jusqu'à présent nous avons présenté les résultats obtenus en utilisant le modèle delta-lognormal vectoriel en trois dimensions pour reconstruire des mouvements simples et nous avons comparer différents profils de vitesse. Dans cette section, nous allons faire une discussion sur la validité du modèle. De plus, des remarques concernant le bruit de lecture ainsi que la signification des paramètres seront faites.

4.7.1 Validation du modèle

Le modèle proposé correspond à ce qui est observé pour les trajectoires en trois dimensions, ainsi que pour les profils de vitesse, dans le cas de mouvements simples. Celui-ci fonctionne très bien pour les membres supérieurs, mais à une performance beaucoup moindre pour les membres inférieurs. Dans plusieurs cas, la reconstruction des trajectoires est excellente, mais certains mouvements sont plus bruités que d'autres. Comme le bruit est présent dans tout système, il ne devrait pas être un facteur qui influence notre décision quant à la validité du modèle, à moins que celui-ci ne soit très important. Les mouvements analysés ont une longueur allant d'environ 50 cm à 100 cm et une erreur quadratique moyenne de reconstruction ne dépassant pas 0,5 cm^2 pour la majorité des cas. Cette dernière est considérée petite, même si l'effet du bruit est visible à quelques endroits. De plus, l'erreur quadratique moyenne pour la reconstruction des profils de vitesse ne dépasse pas 250 cm^2/s^2 ce qui est assez grand pour des profils dont la vitesse maximum varie entre 250 cm/s et 600 cm/s. Cependant, l'erreur est augmentée artificiellement, dans les deux cas, à cause du bruit en fin de signal. Le bruit n'est donc pas significatif et, parce que l'erreur quadratique moyenne est petite si nous prenons en compte les artefacts qui l'augmente artificiellement, nous pouvons dire que le modèle est valide.

Il semble cependant que, d'après les résultats obtenus, le modèle ne s'appliquerait pas à certains mouvements des membres inférieurs. Cette affirmation est appuyée par le fait que les membres inférieurs ont des groupes musculaires différents de ceux des membres supérieurs. En effet, comme le modèle est basé, à l'origine, sur le mouvement des bras, ceci pourrait expliquer la mauvaise performance pour la reconstruction des trajectoires des membres inférieurs.

Cependant, malgré l'argument précédent, nous croyons que le modèle s'applique aux membres inférieurs et que les mauvais résultats obtenus ont été causés par des conditions expérimentales difficiles. En effet, nous avons remarqué que les sujets semblaient perdre l'équilibre lors de certains mouvements des jambes, ce qui entraînait des mouvements involontaires de ceux-ci pour leur permettre de ne pas tomber. Ce phénomène expliquerait le fait que, pour les mouvements des membres inférieurs, nous observions des profils de vitesse qui correspondent à des mouvements complexes. Pour confirmer cette hypothèse, il faudrait s'assurer que le sujet ne sera pas en déséquilibre lors des expérimentations. Cette hypothèse nous semble cependant probable, car pour des mouvements provoquant peu de perte d'équilibre, comme le lever de la jambe vers l'arrière, les courbes observées sont conformes au modèle.

Le modèle est donc valide pour les membres supérieurs et il faudra faire d'autres expérimentations pour ceux inférieurs. Les paramètres que nous avons extraits permettaient de reconstruire les courbes observées, mais ne sont pas assez précis pour faire des comparaisons à moins que celles-ci ne soient grossières comme dans le cas de la courbure. Premièrement, pour la reconstruction d'un mouvement en trois dimensions, plusieurs solutions sont possibles et la solution choisie dépend de l'extracteur et des méthodes utilisés. Des paramètres très différents pourraient donner une solution semblable.

4.7.2 Analyse des sources de bruit de lecture

Nous avons aussi fait une approximation lors de l'analyse des résultats. La capture s'est fait à une fréquence de 97,9 Hz alors que tous les outils utilisés supposaient une fréquence de 100 Hz. Il va de soi que les paramètres ont été affectés par cette différence. Cependant, cette différence, ayant affecté tous les calculs de la même manière, le résultat final est à un facteur d'échelle près de la véritable solution. Notre modèle étant capable de s'adapter pour les changements d'échelle, une différence entre les deux fréquences ne modifie en rien la validité du modèle, mais les valeurs des paramètres sont biaisées.

Le bruit présent dans les données lors de l'acquisition influence aussi les paramètres extraits. Ce bruit peut être causé par plusieurs sources. Dans notre cas, il a été principalement causé par des distorsions du champ magnétique à cause des structures métalliques ainsi que par l'ordinateur présent dans le local d'acquisition. Ces distorsions provoquent des trajectoires plus ou moins courbés ou des vibrations visibles lors d'un déplacement du capteur sur une trajectoire rectiligne. La seule méthode pour éliminer ce type de bruit est de changer l'équipement de local. À défaut d'avoir un local disponible, nous avons été contraints de faire les expérimentations en gardant toujours à l'esprit que nous allions probablement avoir des distorsions dans le signal acquis. Tous les sujets ont donc exécuté les mouvements dans la zone où le signal était le plus stable et les distorsions, minimales. De plus, tout instrument de mesure comporte aussi une erreur de lecture. Celle-ci peut être considérée comme un bruit blanc et la seule manière de diminuer son influence sur les résultats est de filtrer le signal, ce que nous avons fait.

Tous les mouvements exécutés par les humains comportent aussi du bruit qui peut être plus ou moins accentué dépendant du geste. De plus, le corps réagit à un mouvement en appliquant des forces aux bons endroits pour nous permettre de garder l'équilibre. Pour réduire l'effet de la perte d'équilibre, il faudrait maintenir le corps dans une position neutre

et stable quelle que soit le geste exécuté. De l'équipement est donc nécessaire, mais n'étant pas à notre disposition, nous avons dû tenir en compte cette source d'erreur. Pour l'autre type de bruit, provenant de l'activité musculaire, celui-ci diminue lorsqu'un geste est répété assez souvent, car le corps fini par le maîtriser. Les volontaires avaient le droit de répéter le geste jusqu'à ce qu'ils se sentent confortables, mais il semble que pour certains mouvements cette période d'apprentissage n'ait pas été suffisante. Ceci est très visible lorsque l'on voit des gestes dont la forme du profil de vitesse varie beaucoup d'un essai à l'autre.

Pour tenter de déterminer les déplacements involontaires du sujet, nous avons installé deux capteurs en plus de celui utilisé pour l'acquisition du mouvement. La position de ceux-ci dépendait du geste effectué. Ces capteurs n'ont pas été utilisés lors de l'analyse des résultats, car les données recueillies n'étaient pas significatives. Le positionnement n'était pas très précis et les capteurs se déplaçaient sur le corps. Les déplacements étaient causés par le mouvement créé par la contraction des muscles. Comme ce phénomène causait beaucoup de bruit sur le signal que nous tentions d'acquérir, nous avons du abandonner l'idée d'utiliser l'information de ces capteurs pour les analyses.

Cette section a présenté une discussion sur la validité du modèle. Nous avons aussi énuméré les principales sources de bruit, et avons expliqué comment le réduire lorsque c'était possible.

4.8 Conclusion

Dans ce chapitre nous avons présenté le matériel utilisé ainsi que le traitement des résultats. Pour l'acquisition nous avons utilisé l'appareil "A Flock of Birds" de la compagnie Ascension. Nous avons élaboré un protocole expérimental comportant 24 mouvements simples qui étaient exécutés par différentes personnes. Deux personnes se

sont portées volontaires pour notre expérience. Les données ont ensuite été converties par divers logiciels pour que l'on puisse les traiter.

La seconde partie de ce chapitre a montré les principaux résultats obtenus. Nous avons tout d'abord présenté des gestes simples en trois dimensions qui ont été reconstruits à l'aide du modèle delta-lognormal. Des cas où la reconstruction donne d'excellents résultats ainsi que d'autres un peu problématiques ont été montrés et interprétés. Nous avons aussi comparé différentes personnes et différents gestes pour voir s'il y avait possibilité d'utiliser le modèle pour différencier des individus, par exemple. Les résultats recueillis ne nous permettent pas de tirer de conclusions, mais nous ont quand même permis de faire quelques observations intéressantes.

Chapitre 5

Conclusion générale

5.1 Introduction

Le but de ce mémoire était de proposer un modèle pour les mouvements en trois dimensions en s'inspirant du modèle delta-lognormal vectoriel en deux dimensions. Nous y sommes parvenus et nous avons validé ce modèle pour les mouvements des membres supérieurs. Tout nous laisse croire qu'il s'appliquera aussi pour les membres inférieurs. Pour atteindre nos objectifs, plusieurs étapes ont été nécessaires. Nous allons donc présenter une synthèse des travaux réalisés suivi d'une critique de ceux-ci. Nous terminerons en proposant des nouvelles avenues de recherche dans le but de vérifier les domaines où le modèle pourrait être utilisé ainsi que des applications possibles de celui-ci.

5.2 Synthèse

Pour débuter, nous avons fait une recherche bibliographique pour recenser les modèles déjà existants pour étudier les mouvements humains. Nous avons classifié ceux-ci en trois catégories, soient ceux qui modélisent les mouvements en une, deux et trois dimensions. Le modèle delta-lognormal fait partie de la première catégorie. Déjà à cette étape des modèles furent mis de côté, car leur performance étaient inférieure à d'autres.

Pour la modélisation du profil de vitesse, Alimi (1995) ainsi que Plamondon et al. (1993) ont démontré, que le modèle delta-lognormal était supérieur aux autres. Comme le passage en trois dimensions de tous ces modèles passe par l'ajout de paramètres et que ces

paramètres sont indépendants du modèle choisis, nous avons décidé de garder celui qui avait la performance optimale.

Pour les modèles en deux dimensions, la plupart comportaient beaucoup de paramètres. De plus, certains avaient une performance qui dépendait du geste exécuté. Nous les avons quand même gardé pour voir quelle extension pouvait s'appliquer à chaque modèle. Les deux modèles qui étaient déjà en trois dimensions n'étaient pas intéressants non plus. Le premier utilisait les équations de Frenet pour reconstruire les mouvements et l'autre, des réseaux neuronaux.

Le modèle delta-lognormal en une et deux dimensions a ensuite été présenté. Celui-ci s'appuie sur la physiologie humaine pour expliquer les courbes de vitesse observées de manière mathématique. Il est basé sur l'interaction de sous-systèmes qui composent deux systèmes principaux, un système agoniste et un antagoniste. Il prend en compte que les deux systèmes travaillent en collaboration et créent une synergie. Le modèle delta-lognormal comporte sept paramètres.

Le modèle en deux dimensions comporte neuf paramètres soit deux de plus que celui à une dimension. Cet ajout prend en compte l'aspect vectoriel d'un geste et est nécessaire pour reconstruire la trajectoire observée lors d'une superposition de gestes simples. Ce modèle a été appliqué à l'analyse de signature entre-autre et donne d'excellents résultats en ce qui concerne la reconstruction de celles-ci. Un extracteur de paramètres a aussi été programmé pour automatiser le travail d'extraction.

Nous avons ensuite proposé des extensions pour généraliser les modèles présentés lors de la revue de la littérature. Les modèles qui ont été rejetés l'étaient pour deux raisons principales, soit qu'ils comportaient trop de paramètres et qu'ils étaient donc trop

complexes, soit pour des raisons de performance. C'est sur cette base que le modèle delta-lognormal a été choisi.

Pour la généralisation en trois dimensions nous avons tout d'abord proposé d'intégrer la torsion, car elle était l'extension logique du modèle en deux dimensions selon les équations de Frenet. Cependant, après avoir analysé cette approche, nous en sommes venus à la conclusion qu'un mouvement simple en trois dimensions est en fait exécuté dans un plan. Nous avons donc défini le mouvement simple comme l'activation d'une synergie, composée de deux systèmes neuro-musculaires, activés par des impulsions synchrones, travaillant dans un plan. Nous avons donc enlevé la torsion comme paramètre intrinsèque du modèle. Cette dernière apparaît lorsque des gestes simples, exécutés dans des plans qui ne sont pas parallèles, se superposent. Elle n'est donc pas, d'après notre modèle et comme nous aurions pu le penser, contrôlée par la personne, mais une conséquence de l'exécution d'une série de gestes simples dans différents plans. Elle dépendra donc du degré de superposition ainsi que de l'orientation des plans.

Le modèle comporte onze paramètres. Les sept premiers (D_1, μ_1, σ_1, t_0, D_2, μ_2 et σ_2) sont ceux de la courbe delta-lognormale qui permet de reconstruire le profil de vitesse. Les deux paramètres suivants (C_0 et θ_0) sont issus du modèle en deux dimensions et permettent de caractériser la courbure de la trajectoire. Les deux derniers paramètres (ρ_0 et ϕ_0) permettent de donner l'orientation du plan contenant le geste simple. Des simulations de gestes simples et complexes ont été présentées à la fin du troisième chapitre.

Le chapitre quatre a passé en revue le matériel ainsi que la procédure d'analyse des résultats utilisée. Un appareil permettant de connaître la position tridimensionnelle de capteurs à l'aide de champs magnétiques nous a permis de faire l'acquisition des différents mouvements nécessaires pour notre analyse. Deux sujets se sont prêtés à notre expérience.

Un total de quarante-huit mouvements ont été exécutés par chaque sujet. Plusieurs outils logiciels ont été conçus et utilisés pour traiter et analyser les données.

Les résultats obtenus sont excellents et démontrent que notre modèle est valide pour les membres supérieurs tant pour la reconstruction du profil de vitesse que pour celle de la trajectoire. Nous avons tout de même observé certains gestes bruités ainsi que d'autres ne correspondant pas au modèle. Il semble, en première analyse, que les profils de vitesse membres inférieurs ne soient pas conforme à ce que nous avions prévu. Il est possible que des phénomènes, comme la perte d'équilibre, soient à l'origine de ces résultats inattendus.

Nous avons aussi montré qu'il y avait des différences entre les différents gestes, les différentes personnes et même lors de répétitions d'un même geste. Ces résultats sont préliminaires et sont issus d'un très petit échantillon, mais nous permettent de croire qu'il serait possible de différencier des gestes ou même d'identifier des personnes. Il faudra faire une étude plus exhaustive pour s'en assurer, mais ceci est hors du cadre de ce projet.

5.3 Critique

5.3.1 Aspects positifs

Dans la section précédente, nous avons fait une synthèse du travail qui a été réalisé au cours de ce projet. Nous sommes maintenant rendus au point où il faut porter un regard critique sur le travail accompli dans le but de proposer des nouvelles avenues de recherche. Un aspect positif de cette recherche est le fait que l'objectif de proposer un modèle en trois dimensions pour les mouvements humains ait été atteint. Ce modèle, en plus de donner d'excellents résultats pour les membres supérieurs, permet de voir les mouvements complexe sous une forme simple. Au lieu d'utiliser des paramètres complexes, par exemple la torsion, le modèle propose d'utiliser une sommation de gestes en deux dimensions. Cette

caractéristique permet ainsi de simplifier l'analyse des gestes et pourrait permettre de mieux comprendre les stratégies de génération des mouvements. C'est une approche élégante qui s'inscrit en continuité avec la simplicité des modèles en une et deux dimensions.

Un autre aspect positif est qu'il nous est possible de d'identifier les membres, jusqu'à un certain point, sans qu'une interface visuelle ne soit nécessaire, à l'aide de l'analyse des courbures. Il faut cependant que les capteurs soient positionnés à des endroits précis sur le sujet. Même si les résultats présentés n'étaient pas très précis, nous avons malgré tout été en mesure d'observer que la courbure dépendait du geste accompli. Il est probable que d'autres paramètres permettent de distinguer les différents membres, mais nous sommes en mesure d'affirmer que la courbure est probablement l'un d'entre eux.

Ce travail a aussi inclus la création d'un logiciel permettant de voir les traces en trois dimensions ainsi que les profils de vitesse associés. Tous les paramètres de chaque courbe delta-lognormale sont modifiables manuellement ce qui permet de tester facilement différentes combinaisons de paramètres. De plus, il est possible d'y ajouter des modules pour extraire les paramètres automatiquement dès que ceux-ci seront disponibles. L'interface et la base de ce programme pourraient donc être réutilisés dans le cadre d'autres recherches à propos du modèle delta-lognormal.

5.3.2 Aspects négatifs

Malgré les points positifs, plusieurs aspects auraient eu avantage à mieux se dérouler. Une première ombre au tableau est au sujet du petit échantillonnage de personnes sur lesquelles nous avons testé le modèle. Il est certain que plus un modèle est testé sur un nombre de personnes important, plus il tend à prouver sa performance. Malheureusement, à cause d'un bris de l'appareil d'acquisition, des tests plus exhaustifs n'ont pu être complétés. Un autre problème est le manque de volontaires. Il est difficile de trouver des

personnes qui ont cinq heures de libres dans une journée surtout s'il est préférable que ces heures soient consécutives. À cause de ce problème et des contraintes d'horaire, seulement deux sujets ont participé à notre expérience.

Un autre problème concernait lui aussi l'expérimentation et, plus particulièrement, les conditions expérimentales. Nous avons fait l'acquisition à l'aide d'un appareil utilisant un champ magnétique pour déterminer la position des capteurs. Faute d'avoir un autre local, nous avons installé l'appareil dans un local où plusieurs structures métalliques sont présentes, ce qui peut causer des erreurs lors de la lecture de la position des capteurs. De plus, nous n'avions pas de matériel pour nous permettre de maintenir les volontaires dans une position stable lors des séances d'acquisition. Ces derniers devaient donc s'efforcer de bouger le moins possible le corps à l'exception du membre qui devait faire le mouvement. Il est difficile de réussir un mouvement sans avoir l'aide de structures ce qui a pu être la cause de certains mauvais résultats observés. Les conditions expérimentales n'étaient donc pas excellentes et gagneraient à être améliorées. Malgré cela, le modèle a permis de reconstruire plusieurs des mouvements observés.

Un dernier aspect négatif est que la performance et le pouvoir explicatif du modèle semble dépendre des membres. Comme nous voulions faire un modèle général, il serait ennuyeux qu'il ne s'applique pas au mouvement des membres inférieurs comme semble indiquer les résultats obtenus. Cependant, il est probable qu'avec de l'entraînement, les résultats seraient différents et changent en faveur du modèle. Notre recherche ne nous permet pas, pour l'instant, de vérifier cette affirmation donc le modèle ne pourra être considéré général que lorsque ces tests seront faits. Son application aux membres supérieurs donne toutefois la possibilité d'amorcer de nombreuses études originales.

5.4 Nouvelles propositions

5.4.1 Travail devant être réalisé à la suite de cette recherche

Les critiques, présentées dans la section précédente, faisant état d'aspects positifs, mais aussi de lacunes, le moment est maintenant venu de proposer de nouvelles avenues de recherche ainsi que des domaines d'application pour notre modèle. Dans cette perspective, nous proposons de porter une grande attention à l'endroit où l'appareil d'acquisition sera installé pour s'assurer que l'équipement pour les expérimentations est adéquat. Ceci est essentiel pour la suite des travaux.

La première proposition que nous faisons est la création d'une banque de données de gestes simples et complexes. Cette banque de données devrait être assez volumineuse pour que différentes personnes puissent l'utiliser pour tester de futurs modèles ou vérifier les modèles existants. De plus, elle devrait comprendre autant des sujets ayant différentes maladies qu'en bonne santé. Ceci pourrait permettre de voir si les pathologies peuvent être décelées et identifiées par différents modèles.

Pour que la banque de donnée puisse être représentative et uniforme, il faudra aussi préparer un protocole expérimental strict qui serait suivi pour toutes les acquisitions. Il ne faudra pas oublier d'y inclure une phase d'apprentissage du geste à effectuer pour que la commande puisse être exécutée de manière automatique. De plus, les personnes devront toujours être installées dans la même position pour que les conditions expérimentales soient uniformes.

Si nous voulons être en mesure d'extraire les paramètres sur une grande banque de données, il est impensable de faire tout le travail manuellement. Il reste donc du travail à faire pour rendre l'extraction automatique. Il ne s'agit pas seulement de rendre l'opération

automatique, il faut aussi s'assurer que les valeurs extraites sont consistantes. L'extracteur devra donc être stable, c'est à dire arriver à des résultats semblables pour des gestes semblables, ce qui est un véritable défi, car la solution n'est pas unique. Il est certain que dans des cas extrêmes c'est impossible, mais dans la majorité des cas la règle de la stabilité devra s'appliquer. De plus, pour l'utilisation du modèle dans des systèmes à temps réel il est inacceptable que l'extraction prenne plusieurs minutes, celui-ci devra donc être rapide. L'extracteur pourra être intégré à l'interface déjà programmée ce qui permettra de faire des ajustements et de simplifier la correction des problèmes qui pourraient survenir.

Lorsque nous aurons une grande banque de données ainsi qu'un extracteur automatique de paramètres qui est stable, beaucoup de travaux pourront être entrepris. Nous pourrons vérifier si le modèle s'applique aux mouvements des membres inférieurs, ce qui devrait être assez facile à partir de données recueillies correctement. C'est à ce moment que nous pourrons démontrer que notre modèle est général. Par la suite, il faudra vérifier si le modèle est capable d'expliquer les mouvements complexes comme nous le prétendons.

Des études statistiques des paramètres pourront être entreprises pour analyser leurs propriétés. Il faudra voir si, à l'aide de certains paramètres, il est possible d'identifier une personne, un geste ou un membre. De même, l'influence de certaines maladies neuro-motrices sur ceux-ci pourra aussi être étudiée. Si tel est le cas, plusieurs personnes et groupes de recherche se sont déjà montrés intéressés par un tel modèle. Par exemple dans les domaines de l'animation par ordinateur, de la sécurité informatique et de la santé.

5.4.2 Applications probables du modèle

Un modèle décrivant les mouvements en trois dimensions est intéressant pour l'animation par ordinateur, car il permet de créer des mouvements naturels en n'ayant

besoin que d'un petit nombre de paramètres. En effet, avec un tel modèle nous n'avons qu'à stocker une dizaine de paramètres au lieu de centaines de points pour un geste simple. Il devient donc avantageux d'utiliser le modèle, même si l'espace de stockage est peu coûteux. De plus, il n'est pas nécessaire d'avoir une personne qui fera tous les mouvements nécessaires pour l'animation. Une fois que nous avons caractérisé chacun des membres, il sera possible de générer n'importe quel mouvement en faisant varier les paramètres. Il s'ensuivra une diminution des coûts de production, car l'appareillage pour l'acquisition ne sera plus nécessaire si nous avons déjà les bons modèles. De plus, il ne sera pas nécessaire d'engager un acteur pour mimer tous les gestes du personnage.

Une autre utilité du modèle concerne la sécurité informatique. Nous avons dit que le modèle pourrait permettre de caractériser des personnes et des gestes. Si tel est le cas, à l'aide de paramètres extraits sur des gestes complexes, nous pourrons être en mesure d'authentifier si la personne qui a fait le geste est bien celle qu'elle prétend être. En s'assurant que chaque personne fait un geste qui lui est caractéristique pour l'authentification, nous pourrons utiliser cet outil comme nous utilisons présentement la signature. Cette signature en trois dimensions est avantageuse, car elle ne laisse pas de traces, donc elle est difficile à reproduire. Par contre, les appareils d'acquisition devront diminuer considérablement de prix et il faudra une période d'adaptation pour que l'on puisse éventuellement l'utiliser comme remplacement à la signature dans certaines applications spécifiques.

Du point de vue de la condition physique, une compagnie qui conçoit des appareils de musculation s'est montrée intéressée à notre modèle. Ils veulent l'utiliser pour comprendre comment les gestes sont générés et être en mesure de créer des appareils de musculation plus efficaces. Il est certain que le modèle présenté dans ce mémoire ne s'appliquera pas directement, car celui-ci est conçu pour des gestes rapides auxquels nous n'appliquons pas de contraintes. Comme les appareils contraignent les gestes et que les

personnes font des gestes généralement lents, il faudra l'adapter. Cependant, le modèle proposé sera un point de départ pour l'élaboration de ces modèles dérivés qui prendront en compte, entre autre, l'effet des charges externes ou de la force gravitationnelle.

Notre modèle pourrait aussi servir d'outil de diagnostique pour les maladies neuro-motrices. Comme chaque geste sera caractérisé par un ensemble de paramètres, il sera possible de voir évoluer ceux-ci au cours du temps. En faisant une étude sur une population d'une taille acceptable, nous pourrons analyser les perturbations et les associer à la maladie qui les a créées. Nous pourrions donc détecter des maladies comme le Parkinson, et suivre son évolution. Des exercices qui ciblent certains problèmes précis pourront aussi être élaborés pour permettre de mieux contrôler ses gestes.

Enfin, un groupe de recherche qui travaille sur des implants intra-musculaires s'intéresse à notre modèle. Ce groupe de recherche veut étudier les gestes qui seront générés à la suite d'un certain train d'impulsions. Comme notre modèle prédit que les courbes de vitesse sont en fait les réponses impulsionnelles de la synergie impliquant des systèmes neuro-musculaires en parfaite condition, notre approche répond donc à leur besoin. Il faudra cependant que le modèle soit ajusté pour chaque application, car les différents muscles ont des propriétés différentes. Ce projet pourrait permettre de redonner l'usage des membres aux personnes qui auraient subi un accident qui aurait fait en sorte qu'un organe aurait été paralysé. Nous pourrions même envisager de construire des prothèses provoquant un mouvement naturel en actionnant des moteurs de manière à ce qu'ils aient une réponse cohérente avec le modèle.

Références

ALIMI, M.A. (1995), "Contribution au développement d'une théorie de génération de mouvements simples et rapides: application au manuscrit", Thèse de doctorat, École Polytechnique de Montréal, Canada.

ALIMI M.A. AND PLAMONDON R. (1993a), "Performance Analysis of Handwritten Strokes Generation Models", Proc. Int. Workshop on Frontiers in Handwriting, Buffalo, USA, May, pp. 272-283.

ALIMI M.A. AND PLAMONDON R.(1993b), "Parameter Analysis of Handwriting Strokes Generation Models", Proc. Int. Conf. Handwriting and Drawing, Paris, France, July, pp. 4-6.

ALIMI M.A. AND PLAMONDON R. (1994), "Analysis of the Parameter Dependence of Handwriting Generation Models on Movement Characteristics", in C. Faure, P. Keuss, G. Lorette, and A. Vinter (eds.) Advances in Handwriting and Drawing, Europia, Paris, pp. 363-378.

COSTA, M., CRISPINO, P., HANOMOLO, A. AND PASERO, E. (1997), "Artificial Neural Networks and the Simulation of Human Movements in CAD Environments", IEEE International Conference on Neural Networks - Conference Proceedings, v. 3, pp. 1781-1784.

DENIER VAN DER GON J.J., THURING J.PH., AND STRACKEE J. (1962),"A Handwriting Simulator", Physics in Medicine and Biology, vol. 6, pp. 407-414.

DOOIJES E.H. (1983), "Analysis of Handwriting Movements", Acta Psychologica, vol. 54, pp. 99-114.

EDELMAN S., AND FLASH T. (1987), "A Model of Handwriting", Biological Cybernetics, vol. 57, pp. 25-36.

EDEN M. (1962), "Handwriting and Pattern Recognition", IRE trans. Information Theory, vol. 8, pp. 160-166.

FLASH, T. (1987), "The Control of Hand Equilibrium Trjectories in Multi-Joint Arm Movements", Biological Cybernetics, vol. 57, pp. 257-274.

FLASH T., AND HOGAN N. (1985), "The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model", The Journal of Neuro-science, vol. 5, pp. 1688-1703.

FLASHNER, H., BEUTER, A. AND ARABYAN, A. (1988), "Fitting Mathematical Functions to Joint Kinematics During Stepping: Implications for Motor Control", Biological Cybernetics, vol. 58, pp. 91-99.

GOODMAN, S.R. AND GOTTLIEB, G.L. (1995), "Analysis of kinematic invariances of multijoint reaching movement", Biological Cybernetics, vol. 73, pp. 311-322.

GUERFALI, W. (1996), "Modèle delta lognormal vectoriel pour l'analyse du mouvement et la génération de l'écriture manuscrite", Thèse de doctorat, École Polytechnique de Montréal, Canada

GUERFALI, W., PLAMONDON, R. (1993), "Techniques d'extraction de paramètres pour le modèle delta log-normal (DL)", Ecole Polytechnique de Montreal, EPM/RT-93/27, December 1993, 15 p.

GUERFALI, W., PLAMONDON, R., (1998), " A New Method for the Analysis of Simple and Complex Planar Rapid Movements", Journal of Neuroscience Methods, Elsevier Science Publishers, 82/1, July 1998, pp. 35-45.

GUTMAN S.R. AND GOTTLIEB G.L. (1991), "Exponential Model with Non-linear Time of Reaching Movement: Trajectory Time Profile, Strategies, Variability", Third IBRO World Congress of Neuroscience, Montreal, Canada, August 4-9, 1991, Paper P39.23, p. 262.

GUTMAN S.R., GOTTLIEB G.L. AND CORCOS D.M. (1992), "Exponential Model of a Reaching Movement Trajectory with Nonlinear Time", Comments Theoretical Biology, vol. 2, no. 5, pp. 357-383.

HOGAN, N. (1985), "The mechanics of multi-joint posture and movement", Biological Cybernetics, vol. 52, Pp. 325-332.

HOLLERBACH J.M. (1981), "An Oscillation Theory of Handwriting", Biological Cybernetics, vol. 39, pp. 139-156.

KATAYAMA, M. AND KAWATO, M. (1993), "Virtual trajectory and stiffness ellipse during multijoint arm movement predicted by neural inverse models", Biological Cybernetics, vol. 69, pp. 353-362.

LECLERC, F. (1989), "Validation d'un modèle générateur de vitesse à profil gaussien sur des signatures manuscrites", Mémoire de maîtrise, École Polytechnique de Montréal, Canada

LECLERC, F. (1996), "Modèle de génération de mouvements rapides en représentation de signatures manuscrites", Thèse de doctorat, École Polytechnique de Montréal, Canada

LECLERC, F., PLAMONDON, R., LORETTE, G. (1992), "Des gaussiennes pour la modélisation des signatures et la segmentation des tracés manuscrits", Traitement du Signal, vol. 9, no 4, pp. 347-358.

LEDUC, N. (1998), "Projet Extraction: Extraction des Paramètres sur des Signatures", Projet de fin d'études, École Polytechnique de Montréal, Canada.

LEDUC, N., PLAMONDON, R. (2001a), "Modeling 3D movements with delta-lognormal synergies", Progress in Motor Control III, article soumis.

LEDUC, N., PLAMONDON, R. (2001b), "A new model to study human movements : the three dimensional delta-lognormal model.", 10th Biennial Conference of the IGS, article soumis.

MAARSE, F. (1987), "The Study of Handwriting Movement", PhD Dissertation, University of Nijmegen, The Netherlands, Feb.

MACDONALD J.S. (1964), "Experimental Studies of Handwriting Signals", PhD Dissertation, Mass. Inst. Tech., Cambridge, Sept.

MERMELSTEIN P., AND EDEN M. (1964), "Experiments on Computer Recognition of Connected Handwritten Words", Inform. Contr., vol. 7, pp. 225-270.

MORASSO, P. (1983), "Three Dimensional Arm Trajectories", Biological Cybernetics, vol. 48, pp. 187-194.

MORASSO, P., AND MUSSA-IVALDI, F.A. (1982), "Trajectory Formation and Handwriting: A Computational Model", Biological Cybernetics, vol. 45, pp. 131-142.

MUSSA IVALDI, F.A., MORASSO, P. AND ZACCARIA, R. (1988), "Kinematic Networks", Biological Cybernetics, vol. 60, pp. 1-16.

OKADOME, T. AND MASAAKI, H. (1995), "Kinematical Theory of Human Sequential Movements", Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp. 1193-1198.

PLAMONDON R. (1989), "Handwriting Control: A Functional Model", in M.J. Rodney and Cotterill (Eds.), "Models of Brain Function", Cambridge

PLAMONDON R. (1991), "On the Origin of Asymmetric Bell-Shaped Velocity Profiles in Rapid-Aimed Movements", in G.E. Stelmach and J. Requin (Eds.), "Tutorials in Motor Neuroscience", Kluwer Academic Publishers, pp. 283-295.

PLAMONDON R. (1992a), "A Theory of Rapid Movements", in G.E. Stelmach and J. Requin (Eds.), Tutorials in Motor Behaviour II", Elsevier Science Publishers, The Netherlands, pp. 55-69.

PLAMONDON R. (1992b), "A Model-Based Segmentution Framework for Computer Processing of Handwriting", Proc. 11 Conf. on Pattern Recognition, The Hague, pp. 303-312, Sept.

PLAMONDON , R. (1995a), "A Kinematics Theory of Rapid Human Movements. Part 1: Movement Representation and Generation", Biological Cybernetics, vol. 72, pp. 295-307.

PLAMONDON, R. (1995b), "A Kinematics Theory of Rapid Human Movements. Part II: Movement Time and Control", Biological Cybernetics, vol. 72, pp. 309-320.

PLAMONDON, R. (1998), "A Kinematic Theory of Rapid Human Movements: Part III: Kinetic Outcomes ", Biological Cybernetics, vol. 78, pp. 133-145.

PLAMONDON R., ALIMI M.A., YERGEAU P., AND LECLERC F. (1993), "Modelling Velocity Profiles of Rapid Movements: A Comparative Study", Biological Cybernetics, vol. 69, pp. 119-128.

PLAMONDON, R. AND GUERFALI, W. (1998), "The generation of handwriting with delta-lognormal synergies", Biological Cybernetics, vol.78, pp. 119-132.

PLAMONDON, R. AND LAMARCHE F. (1986), "Modelization of Handwriting: A System Approach", In H.S.R. Kao, G.P. van Galen, R. Hoosain (Eds.) "Graphonomics; Contemporary Research in Handwriting", Elsevier Sci., Amsterdam, North Holland, pp. 169-183.

TAGA, G. (1995), "A model of the neuro-musculo-skeletal system for human locomotion: I. Emergence of basic gait", Biological Cybernetics, vol. 73, pp. 97-111.

UNO, Y., KAWATO, M. AND SUZUKI, R. (1989), "Formation and Control of Optimal Trajectory in Human Multijoint Arm Movement", Biological Cybernetics, vol. 61, pp. 89-101.

WADA, Y. AND KAWATO, M. (1993) "Arm Movement Trajectory Formation by a Neural Network Incorporating Models of Forward and Inverse Dynamics", Systems and Computers in Japan, vol. 24, no. 12, pp. 64-77.

WINTER, D.A. (1979), "Biomechanics of Human Movement", John Wiley and Sons, NY.

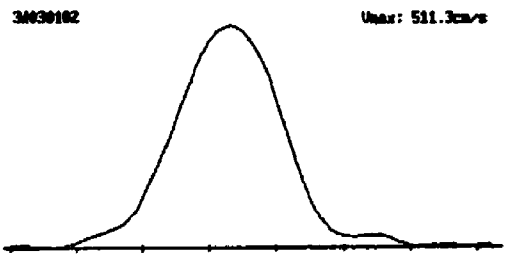
WISE, M.E. (1966), "The Geometry of Log-Normal and Related Distributions and an Application to Tracer-Dilution Curves", Satistica Neerlandica 20, no. 1.

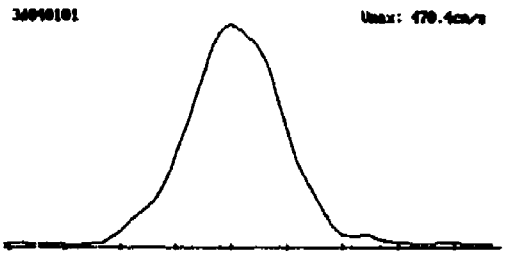
YASUHARA, M. (1975), "Experimental Studies of Handwriting Process", Rep. Univ. Electro. Comm., vol. 25, pp. 233-254.

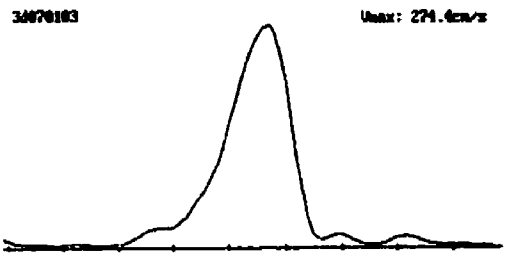
Annexe 1

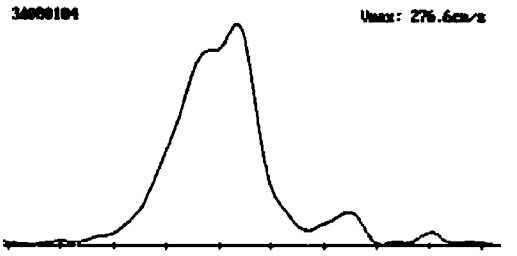
Profils de vitesse

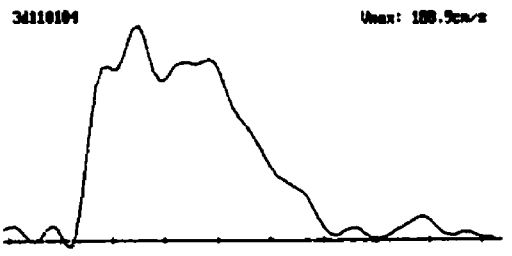
Cette annexe présente les profils de vitesse obtenus lors des expérimentations. Nous avons gardé le profil le plus représentatif pour chaque geste simple. En haut à gauche de chaque profil, le nom du fichier de données correspondant est indiqué. On retrouve la vitesse maximale atteinte en haut à droite. L'axe horizontal est une échelle de temps et chaque graduation représente 0,10 seconde. Ces profils ont été produits, à l'aide des données expérimentale, par le programme écrit par Guerfali (1999).

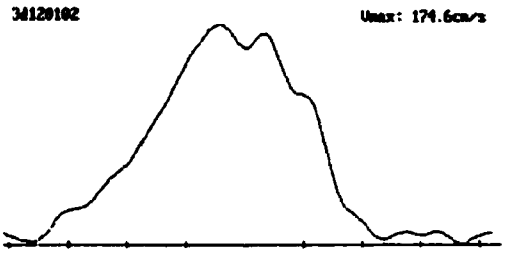

34030102
Umax: 511.3cm/s

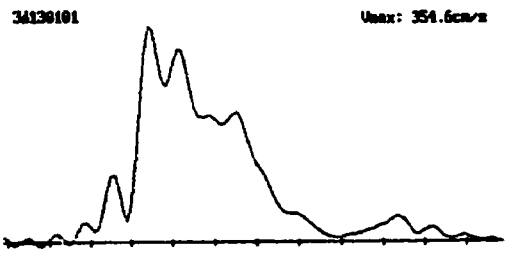

34040101
Umax: 470.4cm/s

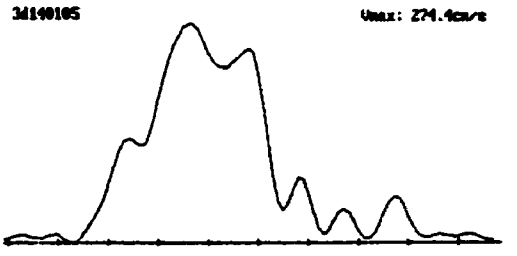

34070103
Umax: 274.6cm/s

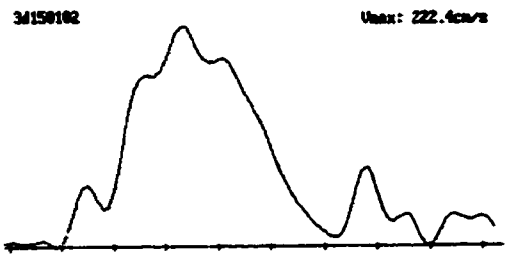

34080104
Umax: 276.6cm/s

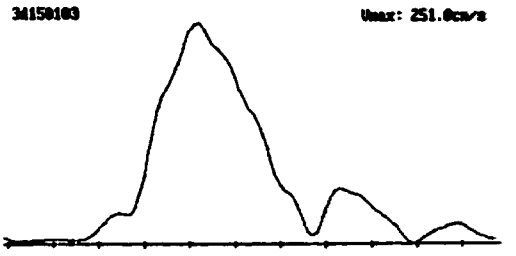

34110104
Umax: 188.9cm/s


34120102
Umax: 174.6cm/s


34130101
Umax: 354.6cm/s

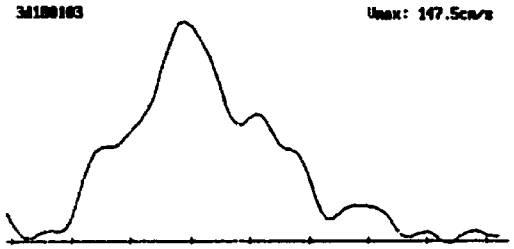

34140105
Umax: 274.4cm/s


34150102
Umax: 222.4cm/s

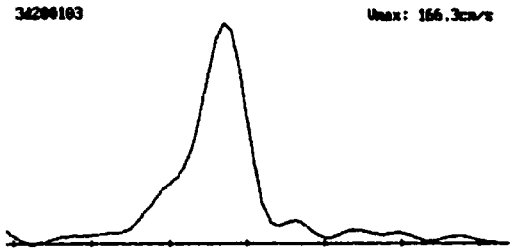

34150103
Umax: 251.0cm/s


34160102
Vmax: 215.0cm/s

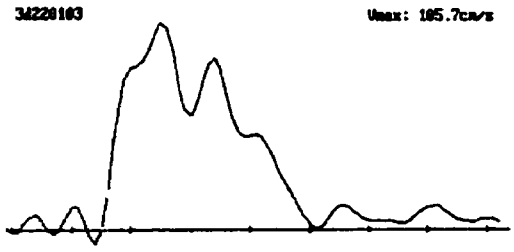

34170101
Vmax: 129.9cm/s

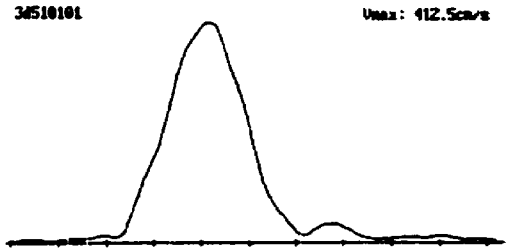

34180103
Vmax: 147.5cm/s

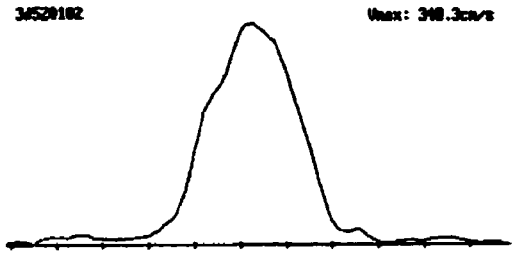

34190101
Vmax: 190.6cm/s

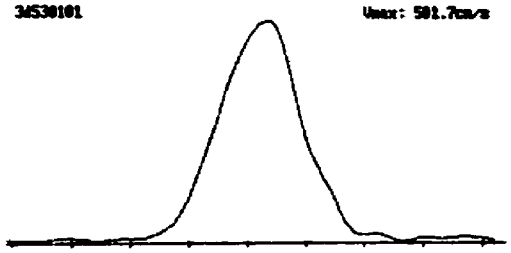

34200103
Vmax: 166.3cm/s

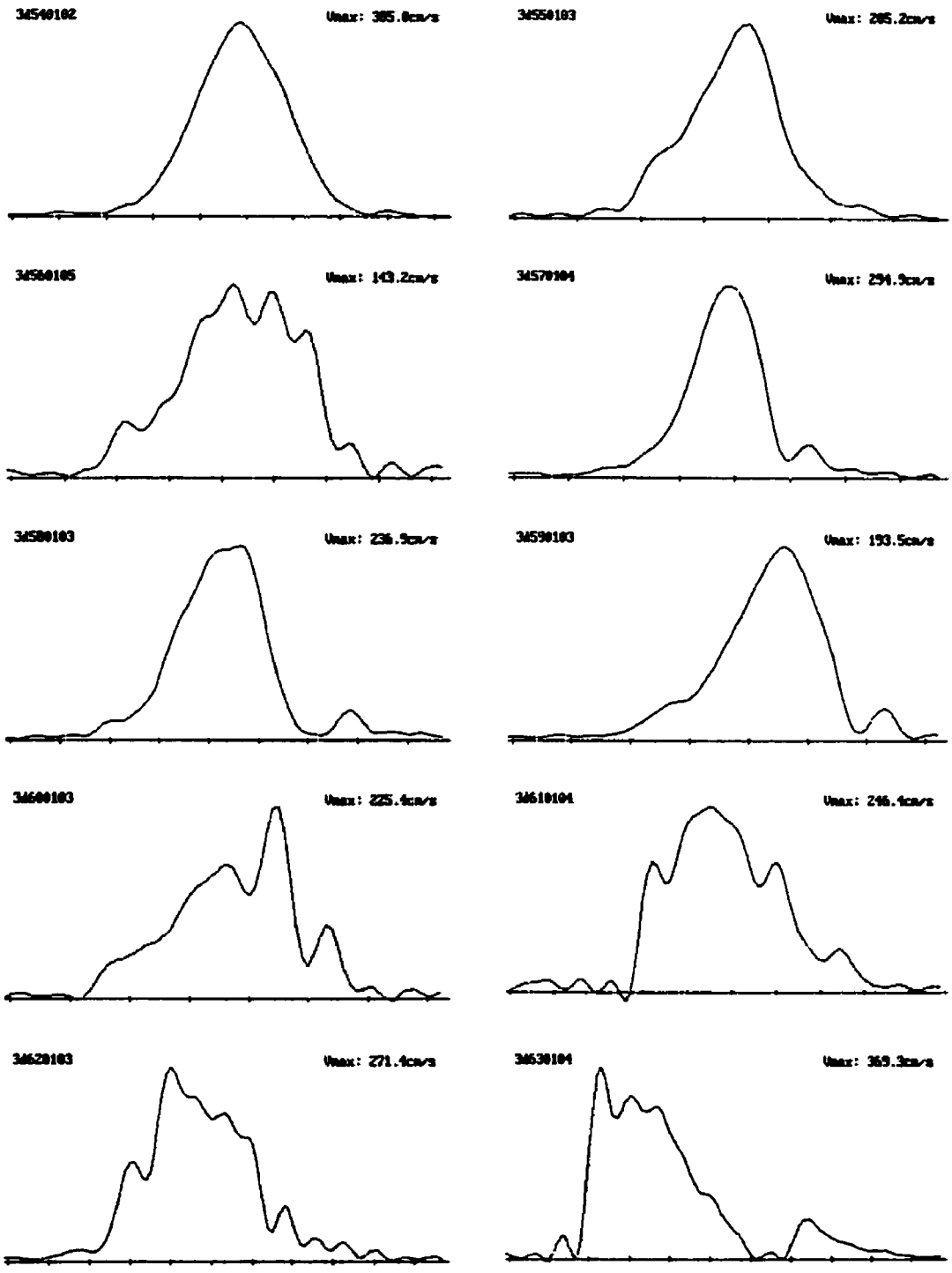

34210102
Vmax: 125.8cm/s

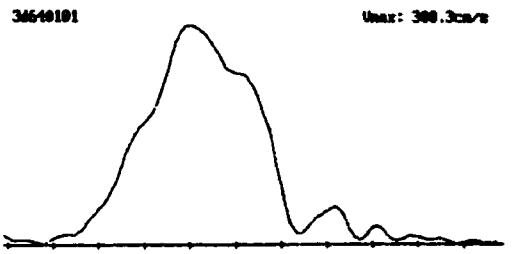

34220103
Vmax: 185.7cm/s

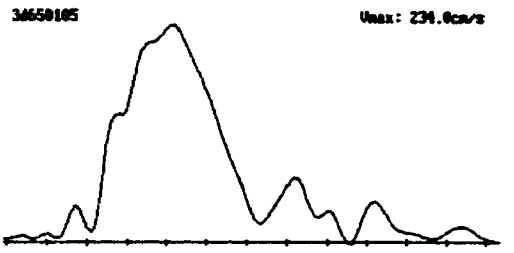

34510101
Vmax: 412.5cm/s

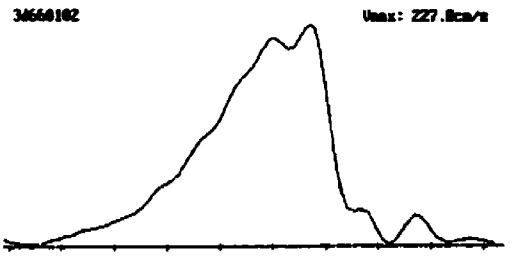

34520102
Vmax: 348.3cm/s

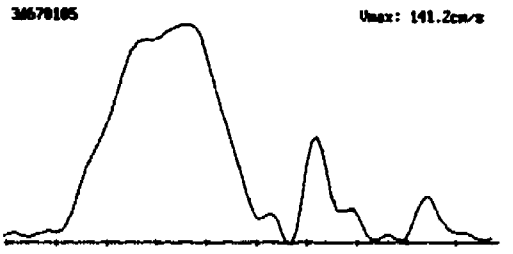

34530101
Vmax: 501.7cm/s

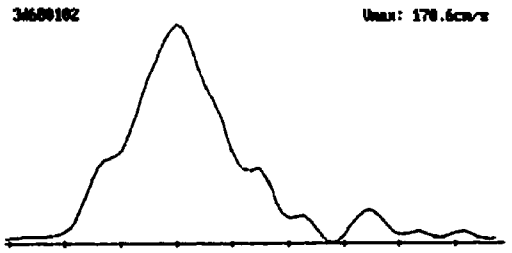

34540102
Vmax: 385.8cm/s
34550103
Vmax: 285.2cm/s
34560105
Vmax: 143.2cm/s
34570104
Vmax: 294.9cm/s
34580103
Vmax: 236.9cm/s
34590103
Vmax: 193.5cm/s
34600103
Vmax: 225.4cm/s
34610104
Vmax: 246.4cm/s
34620103
Vmax: 271.4cm/s
34630104
Vmax: 369.3cm/s

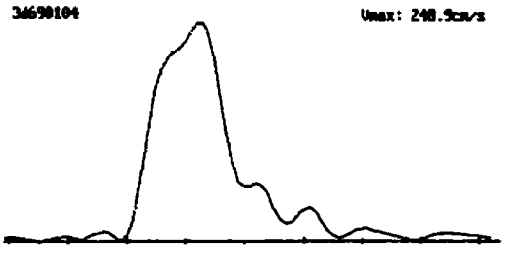

3d640101
Umax: 300.3cm/s

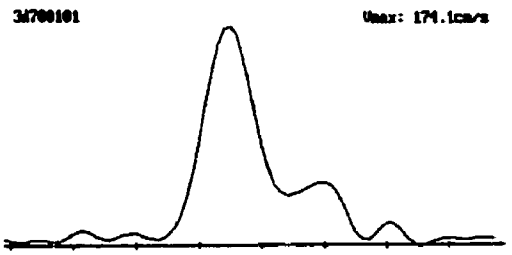

3d650105
Umax: 234.0cm/s

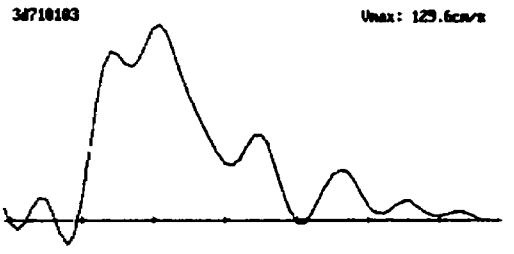

3d660102
Umax: 227.8cm/s

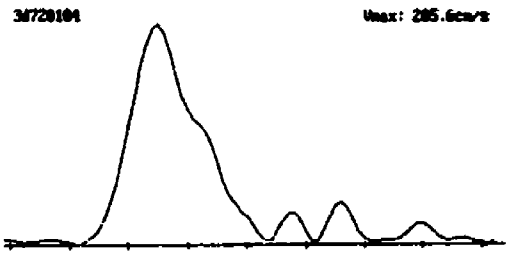

3d670105
Umax: 141.2cm/s

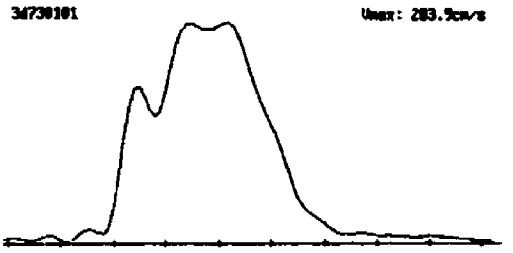

3d680102
Umax: 170.6cm/s

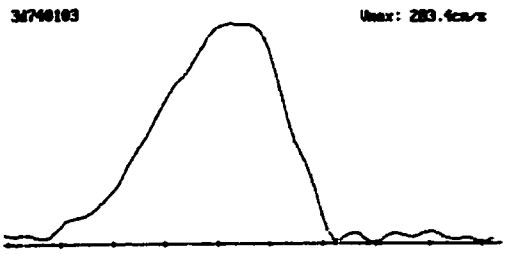

3d690104
Umax: 248.9cm/s

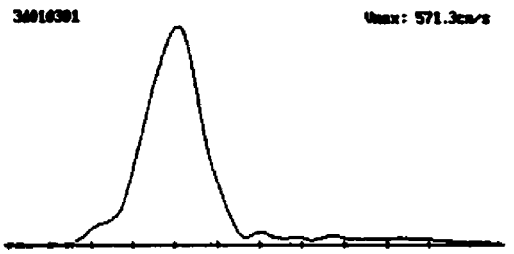

3d700101
Umax: 174.1cm/s

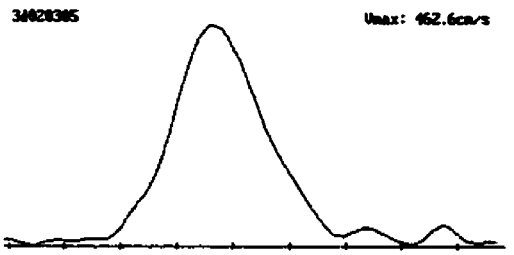

3d710103
Umax: 129.6cm/s

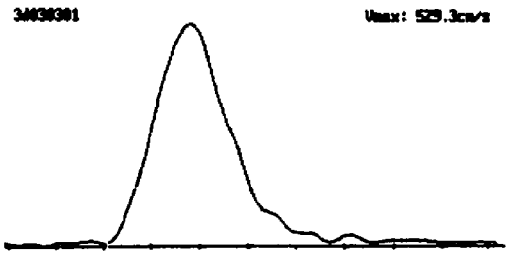

3d720104
Umax: 285.6cm/s

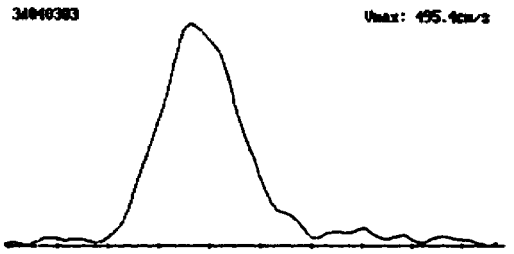

3d730101
Umax: 283.9cm/s


Vmax: 283.4cm/s

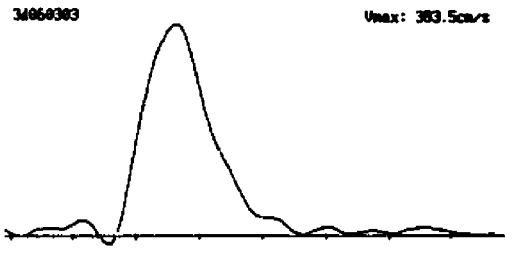

34010301
Vmax: 571.3cm/s

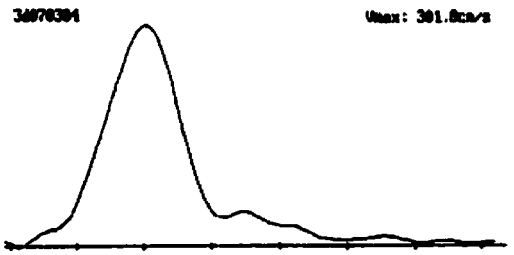

34020305
Vmax: 462.6cm/s

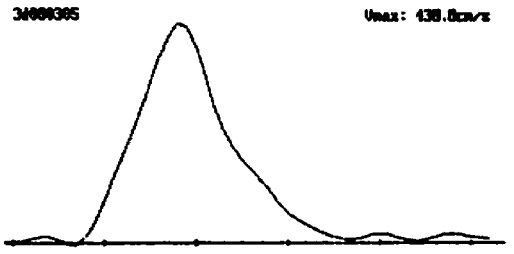

34030301
Vmax: 529.3cm/s


34040303
Vmax: 495.4cm/s


34050302
Vmax: 336.0cm/s


34060303
Vmax: 383.5cm/s

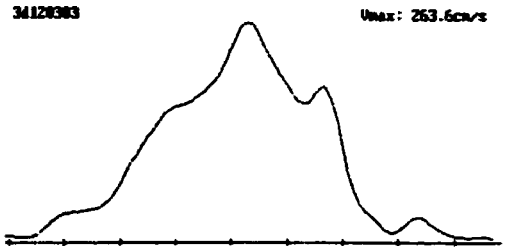

34070304
Vmax: 301.0cm/s


34080305
Vmax: 438.8cm/s

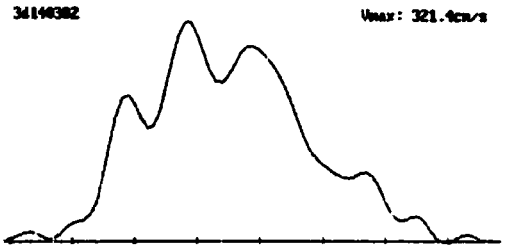

34090301
Vmax: 285.2cm/s

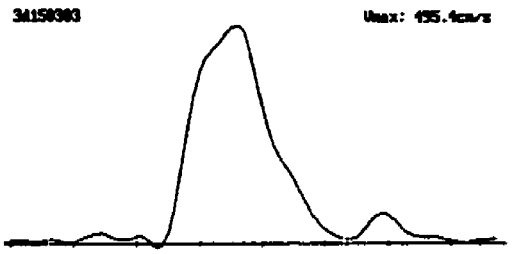

34100305
Vmax: 348.7cm/s

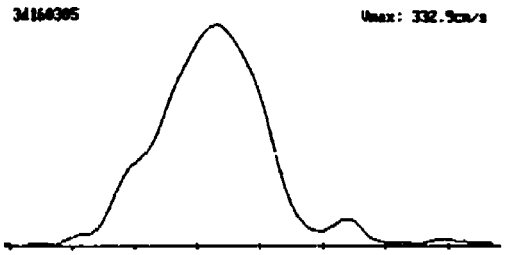

34110301
Vmax: 334.0cm/s

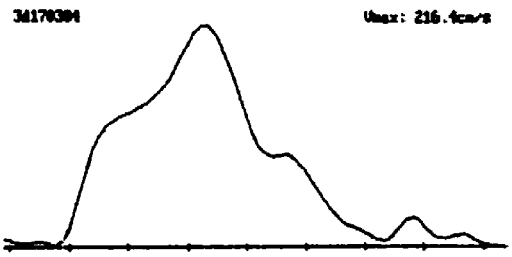

34120303
Vmax: 263.6cm/s

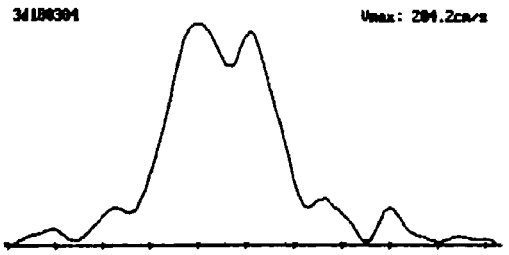

34130305
Vmax: 460.3cm/s

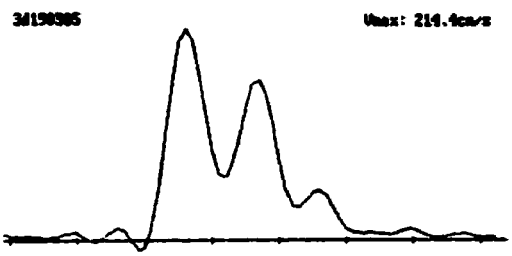

34140302
Vmax: 321.4cm/s

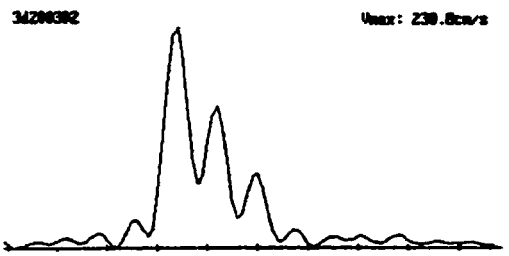

34150303
Vmax: 495.4cm/s

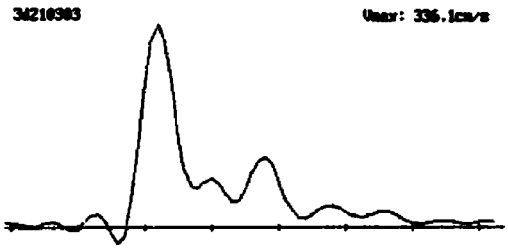

34160305
Vmax: 332.9cm/s


34170304
Vmax: 216.4cm/s

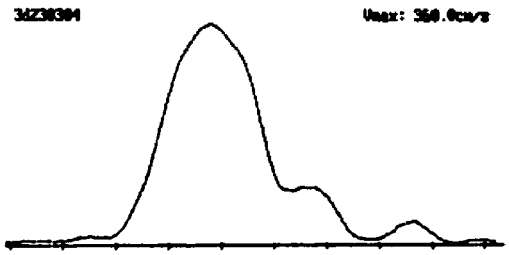

34180301
Vmax: 284.2cm/s

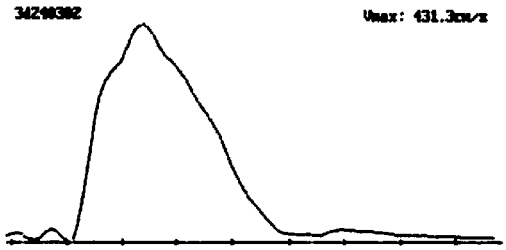

34190305
Vmax: 214.4cm/s

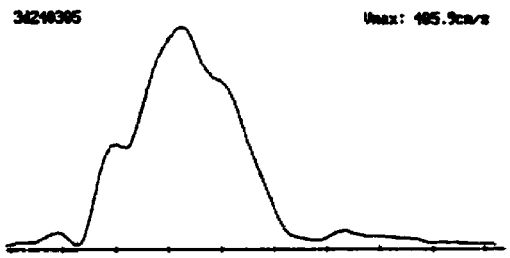

34200302
Vmax: 230.8cm/s


34210303
Vmax: 336.1cm/s

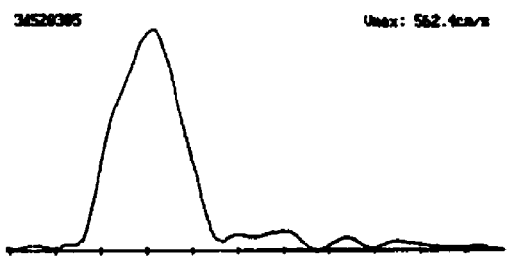

34220305
Vmax: 235.0cm/s

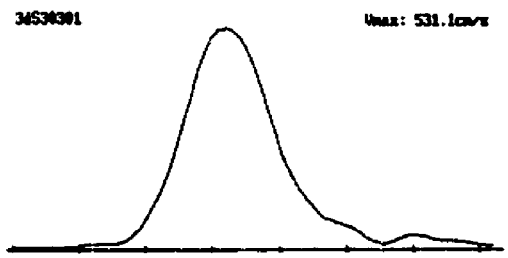

34230304
Vmax: 360.0cm/s

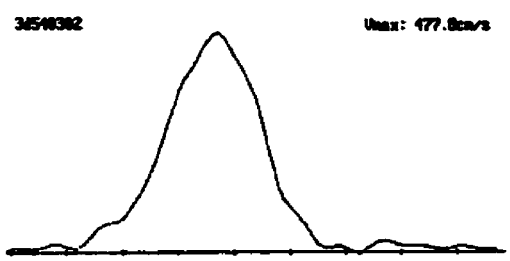

34240302
Vmax: 431.3cm/s

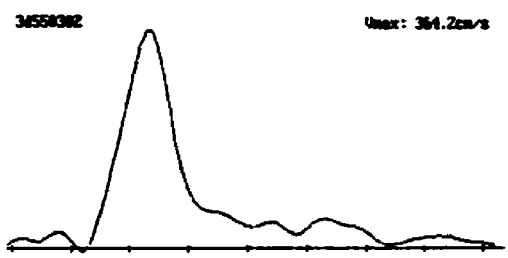

34240305
Vmax: 405.9cm/s

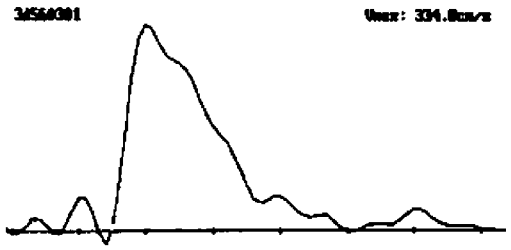

34510305
Vmax: 489.0cm/s

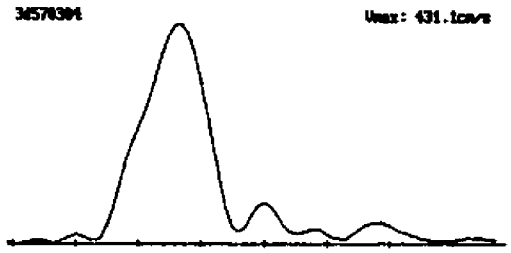

34520305
Vmax: 562.4cm/s

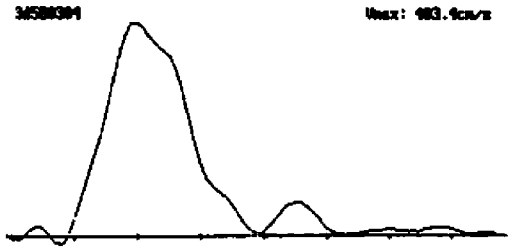

34530301
Vmax: 531.1cm/s

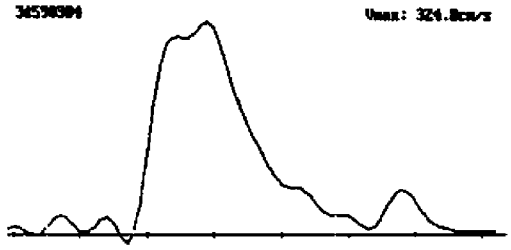

34540302
Vmax: 477.8cm/s

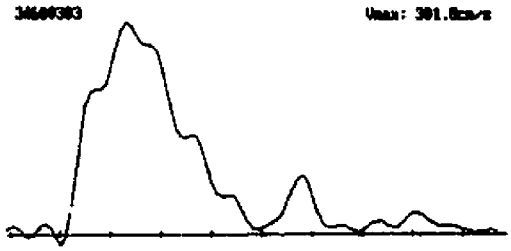

34550302
Vmax: 364.2cm/s

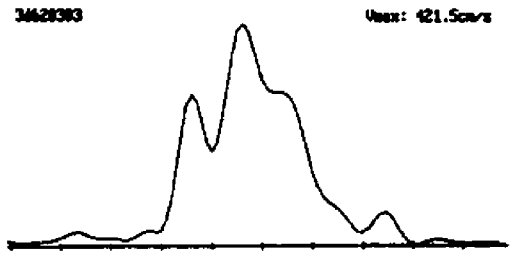


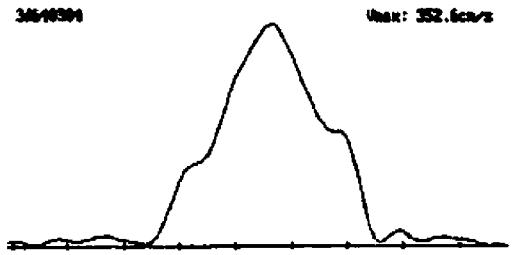


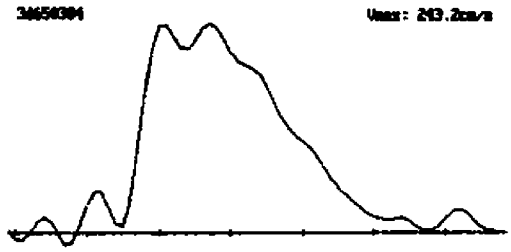


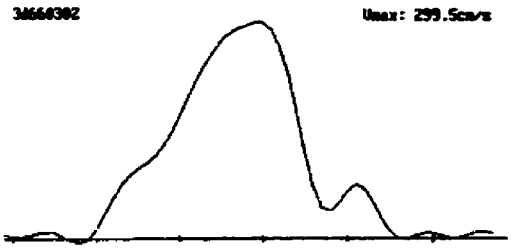


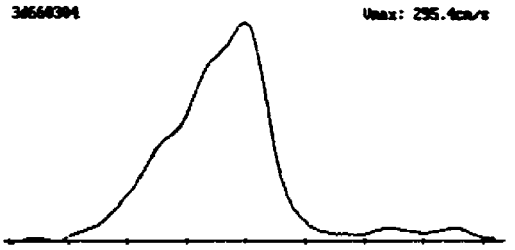




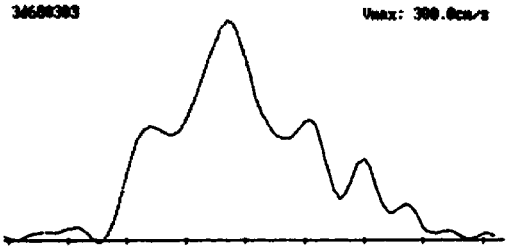


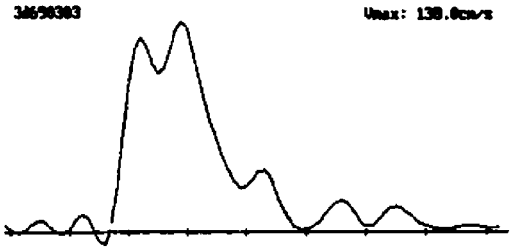


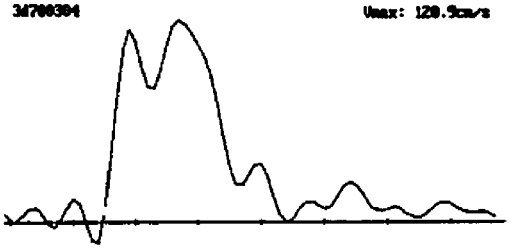

Vmax: 299.5cm/s

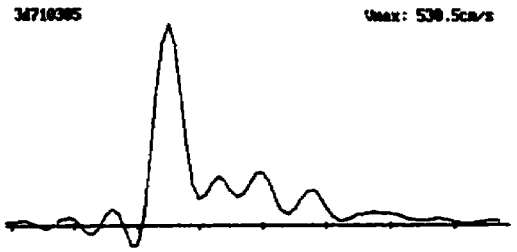

Vmax: 295.4cm/s

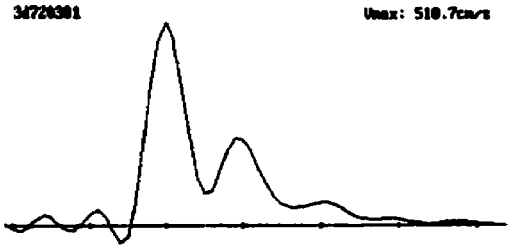

Vmax: 266.3cm/s

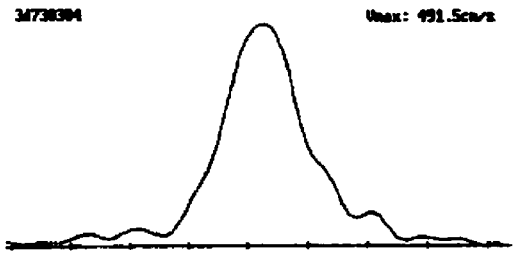

Vmax: 300.0cm/s

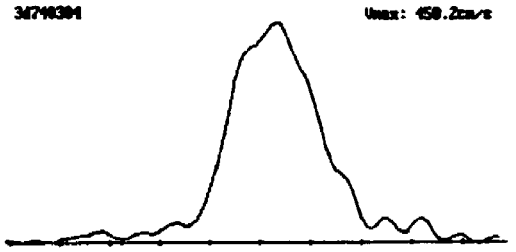

Vmax: 130.0cm/s


Vmax: 120.5cm/s


Vmax: 530.5cm/s


Vmax: 510.7cm/s


Vmax: 491.5cm/s


Vmax: 450.2cm/s

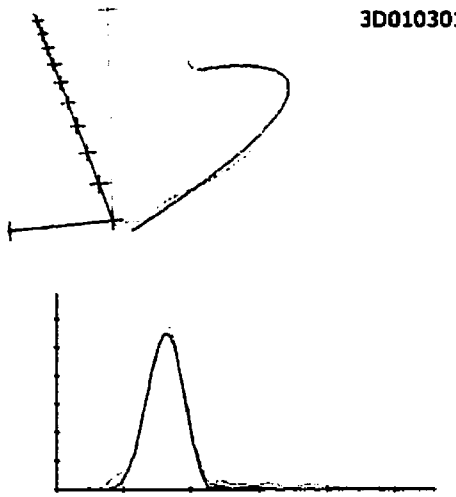
Annexe 2

Reconstruction des trajectoires

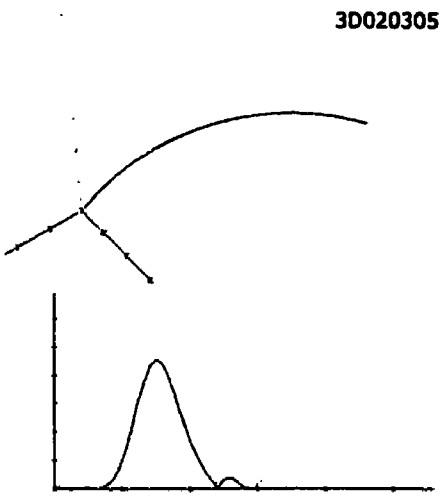
La reconstruction en trois dimensions des profils de vitesse, qui sont présentés dans l'annexe 1 et qui correspondent au modèle delta-lognormal, sont présentés dans cette annexe. On trouve une table présentant les paramètres pour chaque reconstruction présentée ainsi que les erreurs quadratiques moyennes pour le profil de vitesse (EV) et pour la trajectoire (E3D). Les graphiques montrent la trajectoire originale qui n'a pas été filtrée et sa reconstruction. En haut à droite de chacun on retrouve le nom du fichier qui contient les données qui ont permis de générer la trajectoire. Chaque graduation sur les axes de la représentation en trois dimensions représentant 10 cm. Sur axe vertical du profil de vitesse, chaque graduation représente 100 cm/s alors que celles sur l'axe horizontal représentent 0,25 s. Nous avons utilisé le programme conçu dans le cadre de cette recherche par Leduc (2001) pour produire les résultats.

Fichier	D1	Mu1	Sigma1	t0	D2	Mu2	Sigma2	C	Theta	Rho	Phi	x0	y0	z0	E3D	EV
3d010301	149.1875	-0.3441	0.1300	-0.2611	-57.3383	-0.2360	0.0901	1.58	0.582	2.830	1.398	-0.016	-0.004	-0.019	0.0607	211.76
3d020305	91.3661	-0.3244	0.1119	-0.3357	-3.9736	-0.0224	0.0325	1.47	-1.944	-0.213	1.674	-0.006	-0.011	-0.006	0.0070	168.62
3d030102	155.3157	-0.6306	0.2049	-0.1430	-61.4470	-0.4578	0.1308	1.63	0.320	1.541	1.640	-0.016	0.009	-0.009	0.1049	66.09
3d030301	103.5051	-0.2680	0.1005	-0.3766	-7.8703	-0.3835	0.0770	1.60	0.506	1.570	1.578	0.012	-0.004	-0.008	0.4040	121.82
3d040101	172.5177	-0.5670	0.2246	-0.0824	-75.7784	-0.3984	0.1488	1.86	-2.057	-1.318	1.372	-0.002	-0.003	-0.024	0.0785	99.14
3d040303	209.4488	-0.8102	0.2922	0.0259	-109.2591	-0.6557	0.2083	-1.57	-1.006	1.840	1.684	0.017	-0.010	0.007	0.9188	141.08
3d050302	46.4798	-0.8721	0.1359	-0.1550	-7.6748	-0.6006	0.0677	-3.31	0.130	0.883	0.383	0.001	0.003	-0.007	0.0363	138.03
3d060303	48.4252	-1.6651	0.2765	0.0897	-2.4059	-1.0759	0.0506	2.72	1.347	1.462	0.199	-0.009	-0.011	-0.001	0.8323	123.98
3d070103	60.7279	-0.5653	0.1459	-0.0654	-24.7243	-0.4476	0.0793	3.77	0.270	1.361	1.562	0.000	0.008	-0.015	0.0375	83.65
3d070304	58.4273	-0.5196	0.1201	-0.3602	-23.4557	-0.3814	0.0913	3.56	0.454	1.153	1.616	0.008	0.007	-0.008	0.5508	67.38
3d080305	55.0806	-1.0593	0.1491	-0.1450	-3.4838	-0.9515	0.0484	4.42	0.759	1.712	-1.289	0.000	-0.020	-0.003	0.0884	74.59
3d090301	106.5761	-1.4902	0.6245	0.1451	-61.5271	-1.2401	0.4198	3.80	-1.515	-0.117	-1.692	0.008	0.007	-0.003	0.0039	59.83
3d100305	81.2136	-0.8696	0.1956	-0.1271	-43.2174	-0.7405	0.1112	3.26	-0.060	2.850	-1.434	-0.007	-0.006	-0.009	0.0111	129.28
3d150103	62.4233	-0.5477	0.1806	-0.1244	-11.0078	-0.1274	0.0661	1.41	0.689	-1.570	1.164	0.000	0.000	0.000	0.0323	80.84
3d150303	71.9442	-1.4121	0.2406	0.1200	-5.7363	-0.7310	0.0698	0.99	0.871	-1.687	0.935	-0.068	0.007	-0.065	0.5272	227.85
3d160305	104.5519	-0.3136	0.1649	-0.3447	-40.6917	-0.1670	0.0906	1.31	-1.158	2.275	1.535	0.000	0.000	0.000	0.0920	52.22
3d190101	35.8840	-1.0596	0.2058	-0.0262	-16.5744	-0.9126	0.1037	16.25	-0.454	1.785	-1.534	0.003	0.000	-0.001	0.1003	20.49
3d230304	99.0298	-1.1070	0.3597	0.0986	-31.4732	-0.7265	0.1155	-1.92	1.429	0.056	-0.116	0.000	-0.012	0.005	0.4603	208.64
3d240302	174.3545	-1.2565	0.8509	0.1075	-68.2281	-0.7625	0.3551	2.53	-1.736	-1.807	-0.239	0.000	-0.008	0.010	0.2206	105.87
3d510101	104.9259	-0.1960	0.1137	-0.3796	-27.9907	-0.0632	0.1014	2.02	0.352	0.282	1.765	0.000	-0.013	0.011	0.3028	65.51
3d510305	227.5377	-0.9417	0.3822	0.0178	-124.4540	-0.7324	0.2666	1.79	0.418	0.282	1.722	0.000	0.018	0.000	0.4680	247.33
3d520102	99.9643	-0.6543	0.2310	0.0468	-20.6509	-0.3697	0.0983	1.65	1.351	0.083	-1.545	0.000	0.012	-0.013	0.0396	67.58
3d520305	144.9900	-1.1751	0.3669	0.0491	-42.5695	-0.8534	0.1509	1.56	1.136	-0.115	-1.566	0.000	0.004	0.000	0.9620	199.10
3d530101	141.6429	-0.4415	0.1495	-0.1661	-55.9018	-0.3254	0.1039	1.57	0.426	1.432	1.408	0.009	0.000	0.002	0.0481	102.81
3d530301	84.8571	-0.5663	0.1136	-0.2313	-10.2484	-2.5593	0.0100	1.61	0.606	1.616	1.481	0.037	0.000	0.000	0.2344	197.59
3d540102	115.9185	0.0941	0.1039	-0.5738	-25.9582	0.2070	0.0821	1.64	-1.849	-1.553	1.616	-0.004	0.000	0.000	0.1552	25.30
3d540302	170.9927	-0.0013	0.1178	-0.5648	-72.7662	0.0823	0.0784	1.70	-1.935	-1.460	1.591	0.001	-0.011	-0.010	0.4939	94.62
3d560301	54.0301	-2.1578	0.7506	0.1525	-9.1561	-1.3544	0.1160	-5.32	0.330	0.173	-0.213	0.000	0.025	-0.014	0.3790	243.60
3d570104	70.6147	-0.5515	0.1585	-0.1413	-28.7385	-0.4150	0.0808	3.59	0.267	1.570	1.465	0.003	0.011	-0.007	0.0180	66.39
3d570304	110.1507	-1.0748	0.2733	-0.0152	-57.3838	-0.9088	0.1464	2.42	0.451	1.564	1.428	0.038	0.000	0.000	0.3577	116.67
3d580304	61.8070	-1.0254	0.1708	-0.1316	-5.4637	-0.5277	0.0536	3.54	1.068	1.645	-1.546	-0.003	-0.014	0.000	0.2450	130.40
3d590103	69.6966	-0.2865	0.1806	-0.2160	-31.7930	-0.1783	0.0862	4.12	1.600	0.537	0.906	0.024	0.006	0.000	0.1639	58.74
3d660302	67.2783	-0.3437	0.1279	-0.4048	-20.6299	-0.2048	0.0496	1.85	1.834	-0.474	-1.337	0.000	-0.009	0.000	0.2907	139.91
3d700101	18.3974	-1.0697	0.1256	0.0188	-5.0518	-0.7374	0.0688	7.63	1.533	1.413	1.843	0.000	0.000	0.000	0.1536	63.33
3d730304	81.0825	-0.6059	0.1240	-0.1068	-5.4999	-0.3216	0.0271	1.93	1.935	-0.332	0.205	0.000	0.000	0.006	0.1147	367.46
3d740103	107.2324	0.9381	0.0583	-2.0984	-23.4064	1.0045	0.0254	-2.24	-0.225	0.403	-0.049	0.000	-0.003	-0.032	0.0974	31.16
3d740304	94.9513	-0.6890	0.1712	0.0390	-5.1065	-0.3374	0.0316	-2.32	-1.467	1.629	0.199	0.000	-0.016	0.011	0.8846	231.07

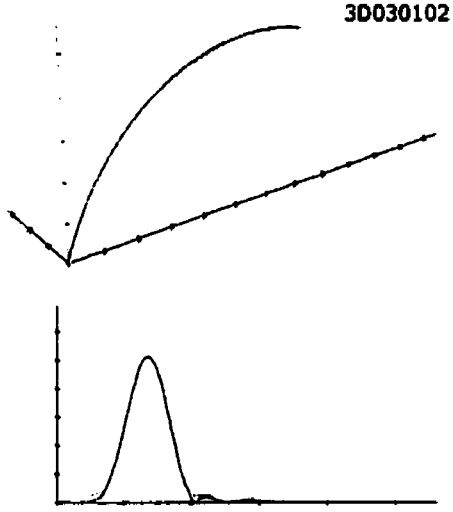
3D010301



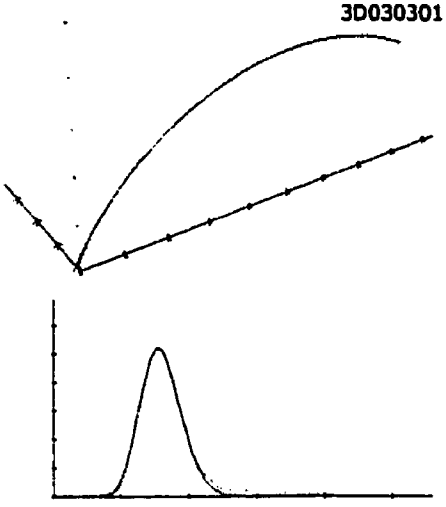
3D020305



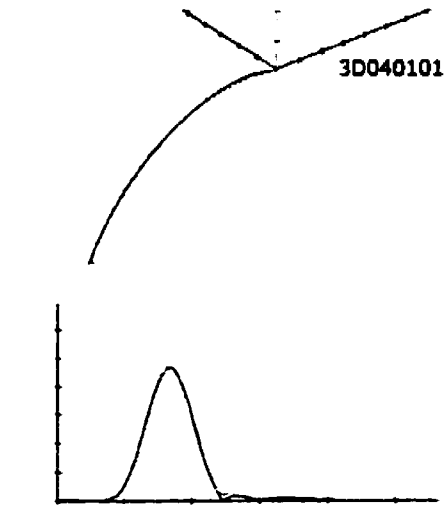
3D030102



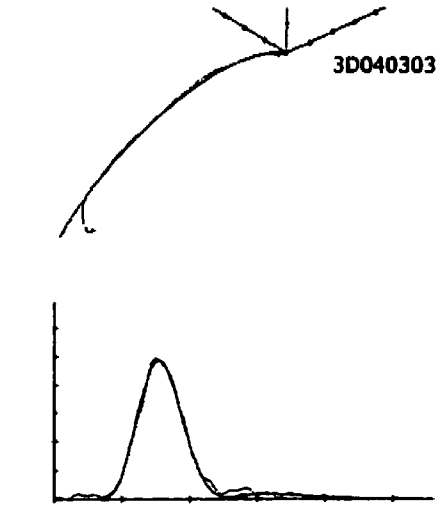
3D030301



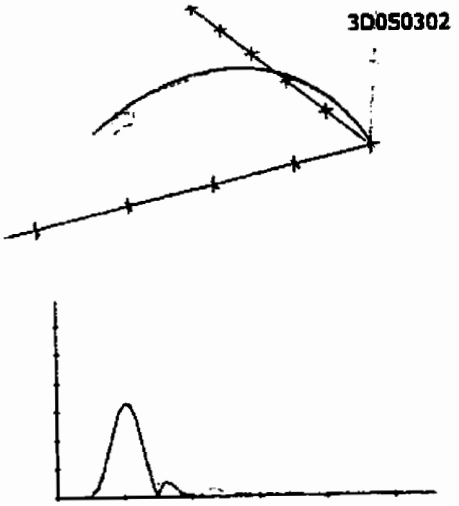
3D040101



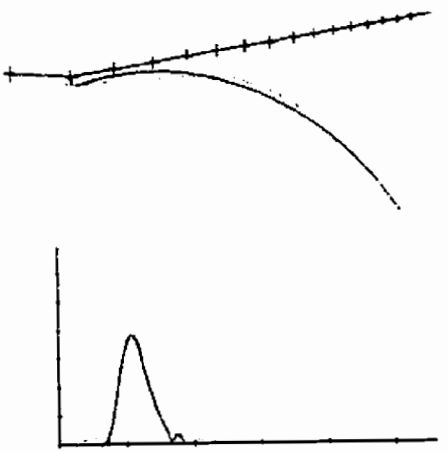
3D040303



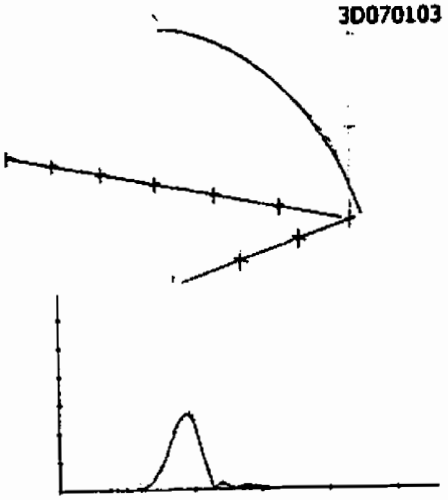
3D050302



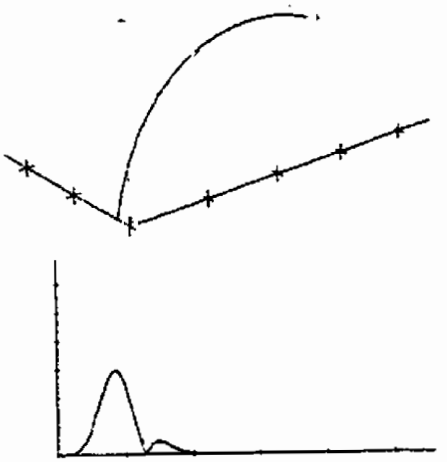
3D060303



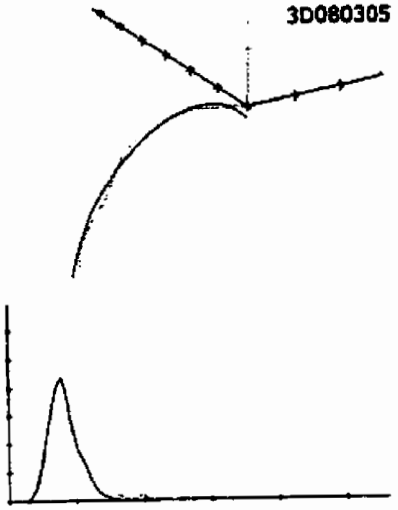
3D070103



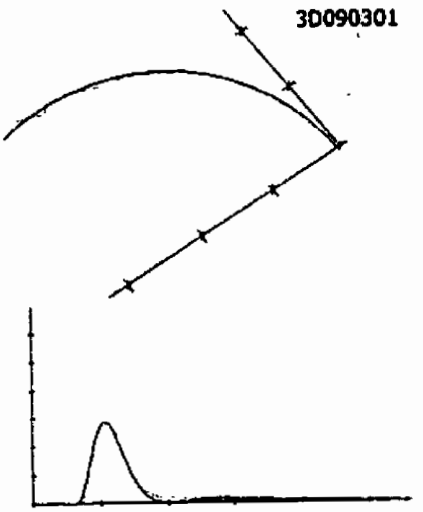
3D070304



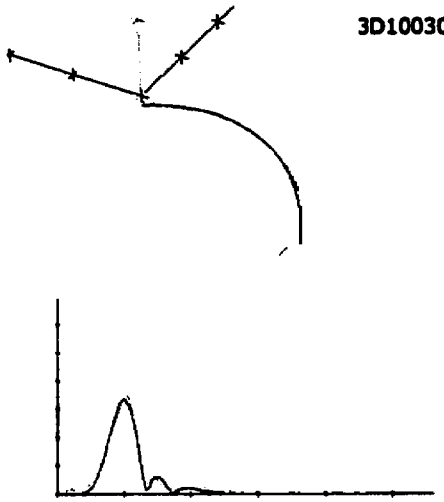
3D080305



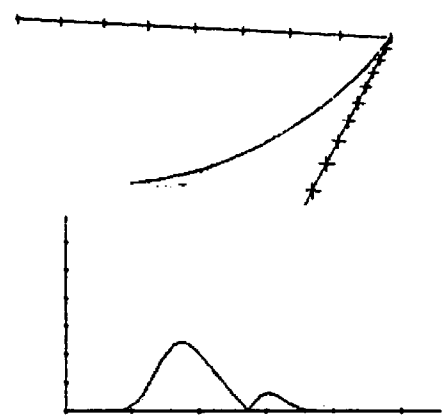
3D090301



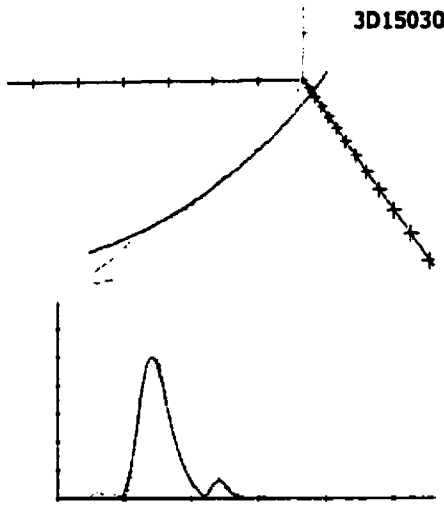
3D100305



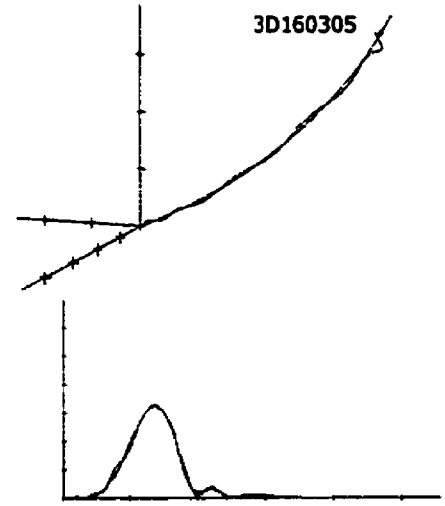
3D150103



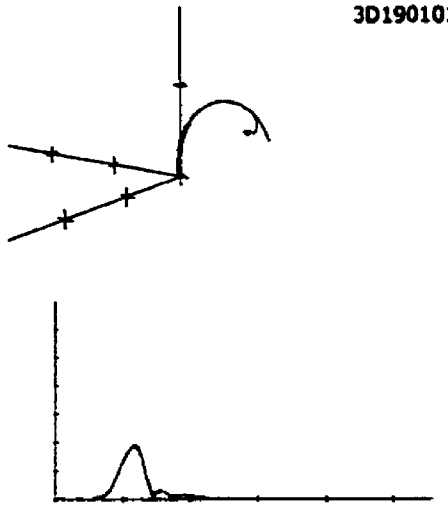
3D150303



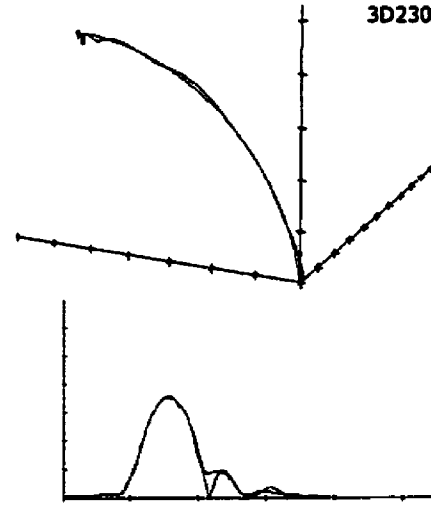
3D160305

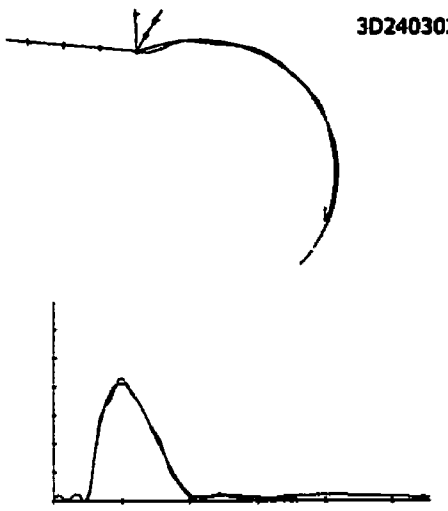


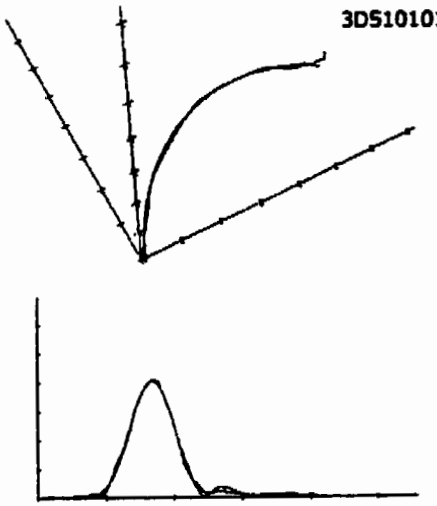
3D190101

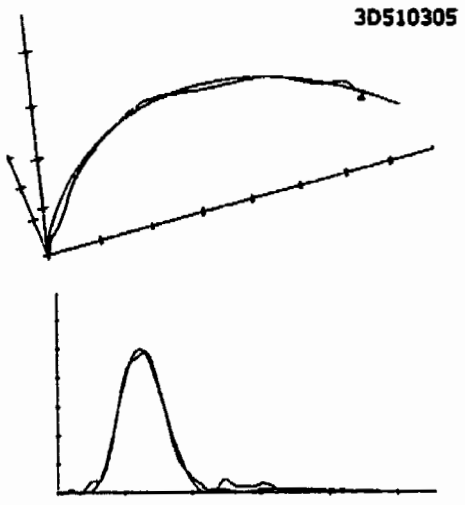


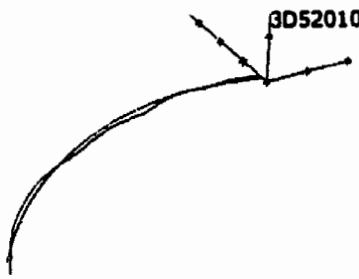
3D230304

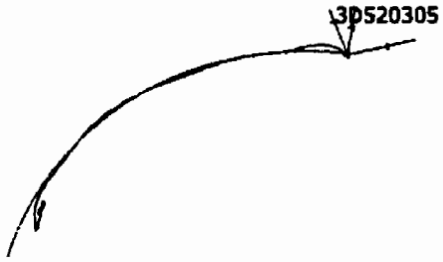


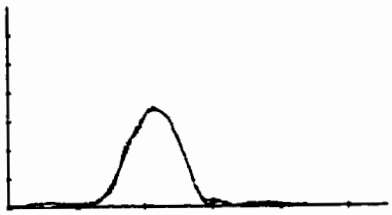

3D240302

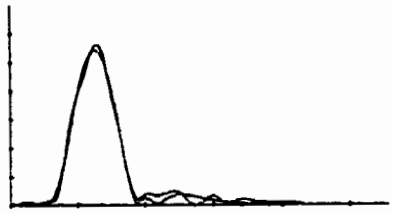

3D510101

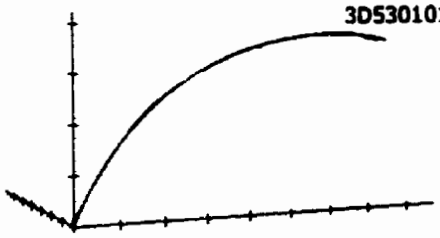

3D510305


3D520102

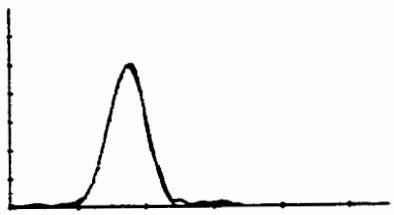

3D520305

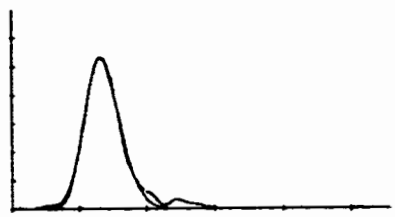


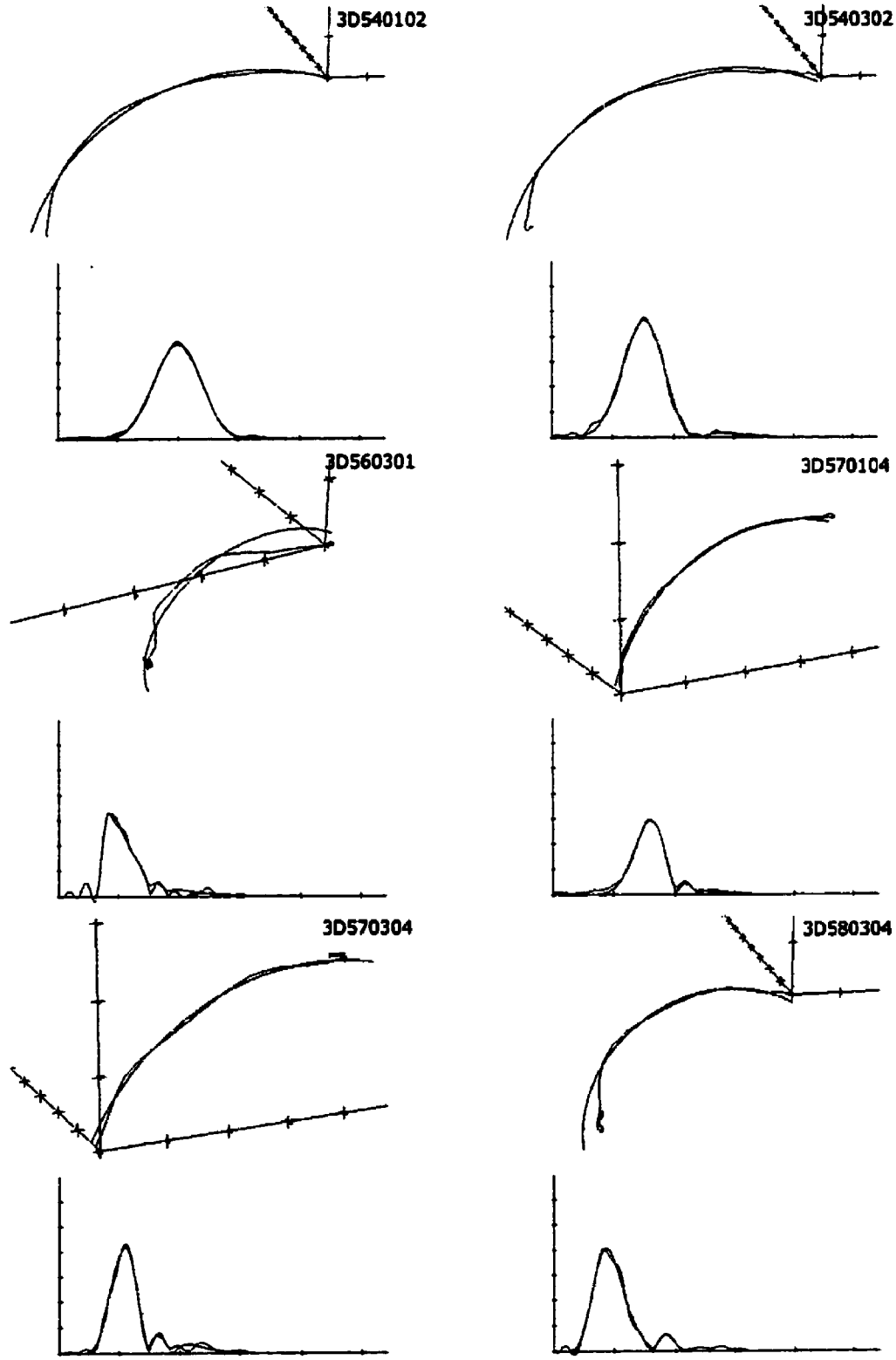


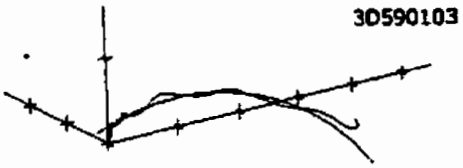

3D530101


3D530301

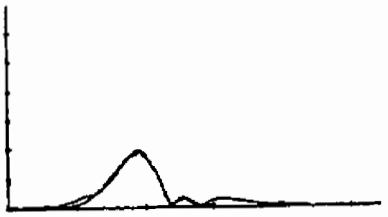


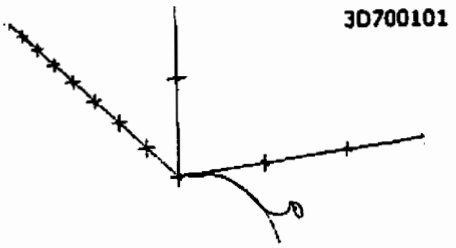


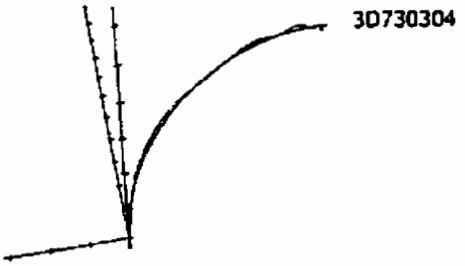

3D540102
3D540302
3D560301
3D570104
3D570304
3D580304

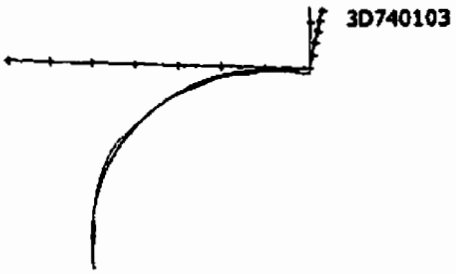

3D590103

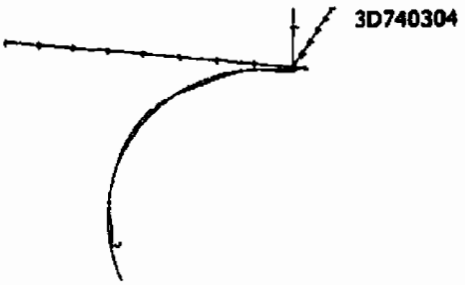

3D660302

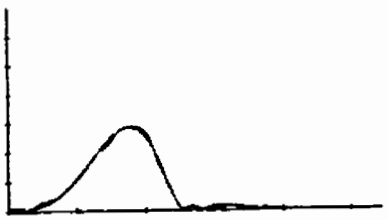


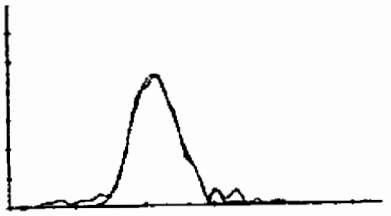

3D700101


3D730304


3D740103


3D740304





Annexe 3

Programme de visualisation des données

Le programme présenté dans cette annexe a permis de trouver les endroits où nous devions séparer les données brutes pour isoler les différents mouvements. Il utilise les librairies graphiques de Borland pour faire l'affichage des courbes de vitesse. Il a été conçu par Leduc (2000) dans le cadre de ce projet. Le code est disponible sur demande par courriel. (hexabit@step.polymtl.ca)

Mar 25, 01 8:43 d:\matrise\doc\annexes\view.cpp Page 1/4

```
//////////////////////////////////////////
// PROGRAMME    Visualiser profils
// BUT          Visualiser les profils de vitesse
// FICHIER      view.cpp
// DESCRIPTION  Permet de visualiser les profils de vitesse
//              et de connaitre la position de curseurs deplacables
// REMARQUE     Utilise les librairies graphiques de Borland
// AUTEUR       Nicolas Leduc
// DATE         15 janvier 2001
//////////////////////////////////////////

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <graphics.h>
#include "bcmath"
#include "umath"

#define ESC 0x1b

void Initialise(void);
void Pause(void);
void Dessine(double *v,int Nb,char nom[]);

//////////////////////////////////////////
// FONCTION     main
// DESCRIPTION  Programme principal
// PARAMETRES   Nom du fichier a visualiser
// RETOUR       Aucun
//////////////////////////////////////////

int main(int argc,char *argv[])
{
  int i,Nb;
  char nom[50];
  double x[17000],y[17000],z[17000],v[17000];

// Si le nom d'arguments n'est pas de 2 donner l'aide
  if (argc<2)
  {
    printf("Usage: view <nom>\n");
    exit(1);
  }
  i=0;
// Copie du nom du fichier dans une variable
  while (argv[1][i]!=0)
  {
    nom[i]=argv[1][i];
    i++;
  }
  nom[i]=0;
// Lecture des positions en X, Y et Z du fichier .RAN
// tel que genere par le programme 4D-Research
  if (!lecture_colonne2(nom,29,1700,10,&Nb,x)==0)
  {
   printf("%s GRRR",nom);
   exit(1);
  }
  if (!lecture_colonne2(nom,29,1700,11,&Nb,y)==0)
  {
   printf("%s GRRR",nom);
   exit(1);
  }
  if (!lecture_colonne2(nom,29,1700,12,&Nb,z)==0)
```

Mar 25, 01 8:43 d:\matrise\doc\annexes\view.cpp Page 2/4

```
  {
   printf("%s GRRR",nom);
   exit(1);
  }
// Calcul de la vitesse  (distance/temps)
  for (i=1;i<Nb;i++)
  {
    v[i]=97.9*sqrt((x[i+1]-x[i])*(x[i+1]-x[i])+(y[i+1]-y[i])*(y[i+1]-y[i])+(z[i+1]-z[i])*(z[i+1]-z[i]));
  }
// Affichage du resultat
  Dessine(v,Nb,nom);
  return(0);
}

//////////////////////////////////////////
// FONCTION     Dessine
// DESCRIPTION  Permet d'afficher les profils de vitesse a l'ecran
// PARAMETRES   v: vecteur vitesse
//              Nb: nombre de points dans le vecteur
//              nom: nom du fichier ;)
// RETOUR       Aucun
//////////////////////////////////////////

void Dessine (double *v,int Nb,char nom[])
{
  char ch;
  int i,curs1,curs2,posx,

  posx=0;
  curs1=0;
  curs2=5;
// Initialise le mode graphique
  Initialise();
// Repete tant que l'usager n'appuie pas sur ESC
  do
  {
// Efface l'ecran
    cleardevice();
// Affiche le profil en blanc
    setcolor(15);
    for (i=1;i<Nb;i++) line(i-posx,480-v[i],i-1-posx,480-v[i-1]);
// Affiche les deux curseurs
    setcolor(13);
    line(curs1-posx,480,curs1-posx,50);
    setcolor(12);
    line(curs2-posx,480,curs2-posx,50);
// Affiche le nom du fichier et les positions des curseurs
    gotoxy(1,1);
    printf("Nom = %s\nCurseur1 = %d\nCurseur2 = %d\n",nom,curs1,curs2);
// Attendre la pression d'une touche
    ch=getch();
    if (ch==0)
    {
// On deplace l'affichage
      ch=getch();
      if (ch==77) posx+=10; //Droite
      if (ch==75) posx-=10; //Gauche
      ch=0;
    }
// Deplacement des curseurs
// Test
    if (ch=='a'&&curs1<Nb-1) curs1++;
    if (ch=='q'&&curs1>posx) curs1--;
    if (ch=='z'&&curs1>1+posx) curs1--
```

Mar 25, 01 8:43 d:\matrise\doc\annexes\view.cpp Page 3/4

```
    if (ch=='m'&&curs2<Nb-1) curs2++;
// Rapide
    if (ch=='Z'&&curs1>50) curs1-=50;
    if (ch=='X'&&curs1<curs2-51) curs1+=50;
    if (ch=='N'&&curs1<$1&curs2) curs2+=50;
    if (ch=='M'&&curs2<Nb-51) curs2+=50;
  }
  while (ch!=27);
  closegraph();
}

//////////////////////////////////////////
// FONCTION     Initialise
// DESCRIPTION  Initialise le mode graphique
//              Routine provenant du programme BGIDEMO.C (Borland)
// PARAMETRES   Aucun
// RETOUR       Aucun
//////////////////////////////////////////

void Initialise(void)
{
  int xasp, yasp;       /* Used to read the aspect ratio*/

  int GraphDriver;
  int GraphMode;
  int ErrorCode;
  struct palettetype palette;
  int    MaxColors;
  int    MaxX, MaxY;
  double AspectRatio;

  GraphDriver = DETECT;       /* Request auto-detection  */

  ErrorCode = registerbgidriver(EGAVGA_driver);
/* report any registration errors */
if (ErrorCode < 0)
{
   printf("Graphics error: %s\n", grapherrormsg(ErrorCode));
   printf("Press any key to halt:");
   getch();
   exit(1); /* terminate with an error code */
}

  initgraph( &GraphDriver, &GraphMode, "" );
  ErrorCode = graphresult();      /* read result of initialization*/
  if( ErrorCode != grOk ){    /* error occurred during init  */
    printf("Graphics System Error: %s\n", grapherrormsg( ErrorCode ) );
    exit( 1 );
  }

  getpalette( &palette );     /* read the palette from board  */
  MaxColors = getmaxcolor() + 1; /* read maximum number of colors*/

  MaxX = getmaxx();
  MaxY = getmaxy();       /* read size of screen     */

  getaspectratio( &xasp, &yasp );   /* read the hardware aspect   */
  AspectRatio = (double)xasp / (double)yasp; /* Get correction factor   */

}

//////////////////////////////////////////
// FONCTION     Pause
// DESCRIPTION  Pause l'affichage en attend qu'une touche soit appuye
//              Routine provenant de BGIDEMO.C (Borland)
```

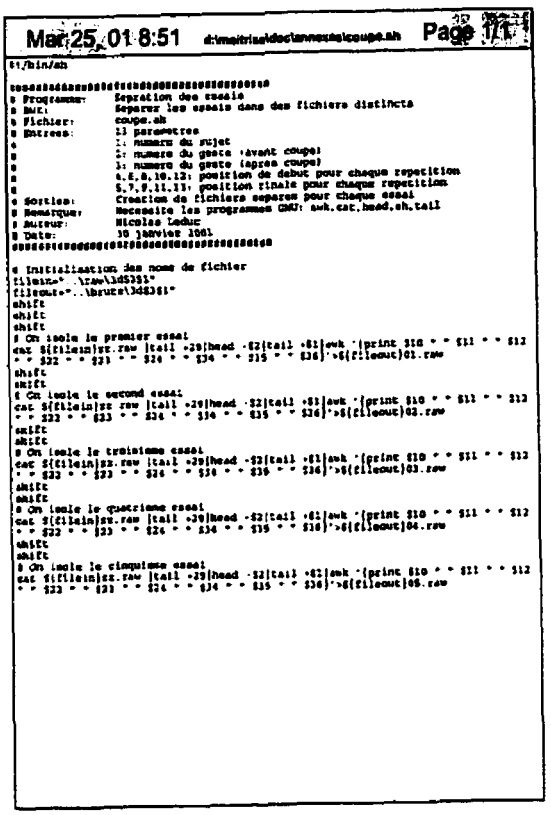
Mar 25, 01 8:43 d:\matrise\doc\annexes\view.cpp Page 4/4

```
// PARAMETRES   Aucun
// RETOUR       Aucun
//////////////////////////////////////////

void Pause(void)
{
  static char msg[] = "Esc aborts or press a key ";
  int c;

  c = getch();          /* Read a character from kbd  */

  if( ESC == c ){       /* Does user wish to leave?   */
    closegraph();       /* Change to text mode        */
    exit( 1 );          /* Return to OS               */
  }

  if( 0 == c ){         /* Did use hit a non-ASCII key? */
    c = getch();        /* Read scan code for keyboard  */
  }

  cleardevice();        /* Clear the screen     */

}
```

Annexe 4

Séparation des gestes

Cette annexe contient le code du script qui a été utilisé pour séparer les fichiers de données brutes en plusieurs fichiers contenant chacun un geste simple. Le script a été écrit par Leduc (2000) et utilise des outils GNU pour traiter les fichiers. Il a été écrit en utilisant le langage du shell sh.



Mar 25 01 8:51 d:\maitrise\doc\annexes\coupe.sh Page 1/1

```
#!/bin/sh

#####################################
# Programme:    Separation des essais
# But:          Separer les essais dans des fichiers distincts
# Fichier:      coupe.sh
# Entrees:      11 parametres
#               1: numero du sujet
#               2: numero du geste (avant coupe)
#               3: numero du geste (apres coupe)
#               4,6,8,10,12: position de debut pour chaque repetition
#               5,7,9,11,13: position finale pour chaque repetition
# Sortie:       Creation de fichiers separes pour chaque essai
# Remarque:     Necessite les programmes GNU: awk,cat,head,sh,tail
# Auteur:       Nicolas Leduc
# Date:         10 janvier 2001
#####################################

# Initialisation des noms de fichier
filein="..\raw\1d5351"
fileout="..\brute\3d8361"
shift
shift
shift
# On isole le premier essai
cat ${filein}yr.raw |tail +29|head -$2|tail +$1|awk '{print $10 " " $11 " " $12
" " $22 " " $23 " " $24 " " $34 " " $35 " " $36}'>${fileout}01.raw
shift
shift
# On isole le second essai
cat ${filein}yr.raw |tail +29|head -$2|tail +$1|awk '{print $10 " " $11 " " $12
" " $22 " " $23 " " $24 " " $34 " " $35 " " $36}'>${fileout}02.raw
shift
shift
# On isole le troisieme essai
cat ${filein}yr.raw |tail +29|head -$2|tail +$1|awk '{print $10 " " $11 " " $12
" " $22 " " $23 " " $24 " " $34 " " $35 " " $36}'>${fileout}03.raw
shift
shift
# On isole le quatrieme essai
cat ${filein}yr.raw |tail +29|head -$2|tail +$1|awk '{print $10 " " $11 " " $12
" " $22 " " $23 " " $24 " " $34 " " $35 " " $36}'>${fileout}04.raw
shift
shift
# On isole le cinquieme essai
cat ${filein}yr.raw |tail +29|head -$2|tail +$1|awk '{print $10 " " $11 " " $12
" " $22 " " $23 " " $24 " " $34 " " $35 " " $36}'>${fileout}05.raw
```

Annexe 5

Conversion des fichiers

Nous avons eu besoin d'un programme qui transforme les données brutes vers un format qui puisse être lu par le programme d'extraction de paramètres et le code de celui-ci est présenté dans cette annexe. Il a été conçu par Leduc (2000) et utilise des routines de Numerical Receipes in C pour calculer les dérivés et filtres le signal. Celles qui ont été utilisées sont présentées dans l'annexe 7. Pour obtenir une copie du code, il faut s'adresser par courriel à hexabit@step.polymtl.ca.

Mar 25, 01 8:59 d:\maitrise\declanneau\obstr.c Page 1/2

```
////////////////////////////////////////
// PROGRAMME      Convertisseur de format
// BUT            Convertir les fichier .RAW en .BRT
// FICHIER        cobrt.c
// DESCRIPTION    Convertit les fichiers creer par le script cmpm.sh
//                (.RAW) en fichier lisable par l'extracteur (.BRT)
// REMARQUE       Aucune
// AUTEUR         Nicolas Lemay
// DATE           30 Janvier 2001
////////////////////////////////////////

#include <stdio.h>
#include <math.h>
#include "fbmath"
#include "lecmask"
#include "mvmlk"

////////////////////////////////////////
// FONCTION       main
// DESCRIPTION    Programme principal
// PARAMETRES     1: Nom du fichier a convertir
//                2: Nom du fichier de sortie
// RETOUR         Aucun
////////////////////////////////////////

int main(int argc,char *argv[])
{
   double *x,*y,*z,*v,*DX,*DY,*DZ,*V,*VF;
   int Nbpt,i;
   FILE *outfic;

// Si nous n'avons pas 3 noms de fichier, on affiche l'aide
   if (argc < 3)
   {
      printf("commande");
      return (0);
   }
// Lecture des position en X,Y,Z
   x=lecture_colonne(argv[1],1,300,1,&Nbpt);
   y=lecture_colonne(argv[1],1,300,2,&Nbpt);
   z=lecture_colonne(argv[1],1,300,3,&Nbpt);
// Derivee dX/dT, DY/DT, DZ/DT
   DX=Filtre_derivatif(x,Nbpt);
   DY=Filtre_derivatif(y,Nbpt);
   DZ=Filtre_derivatif(z,Nbpt);
   v=dvector(1,Nbpt);
   V=dvector(1,Nbpt);
// Calcul de la vitesse et filtrage 'Passe-bas 1dre'
   for (i=1,i<Nbpt;i++)
   {
      v[i]=sqrt(x[i+1]-x[i])*(x[i+1]-x[i])+(y[i+1]-y[i])*(y[i+1]-y[i])+(z[i+1]-z[i])*(z[i+1]-z[i]);
      V[i]=sqrt(DX[i]*DX[i]+DY[i]*DY[i]+DZ[i]*DZ[i]);
   }
   VF=Filtre_pb_1d(V,Nbpt-1);
// Ecriture du fichier de sortie ( BRT)
   outfic=fopen(argv[2],"w+");
   fprintf(outfic,"Data\n");
   fprintf(outfic,"%ldms %ldm",Nbpt-1);
   for (i=1;i<Nbpt;i++)
   {
      fprintf(outfic,"%f %f %f\n",x[i],y[i],z[i]);
   }
   fclose(outfic);
// Liberation des variables de travail
```

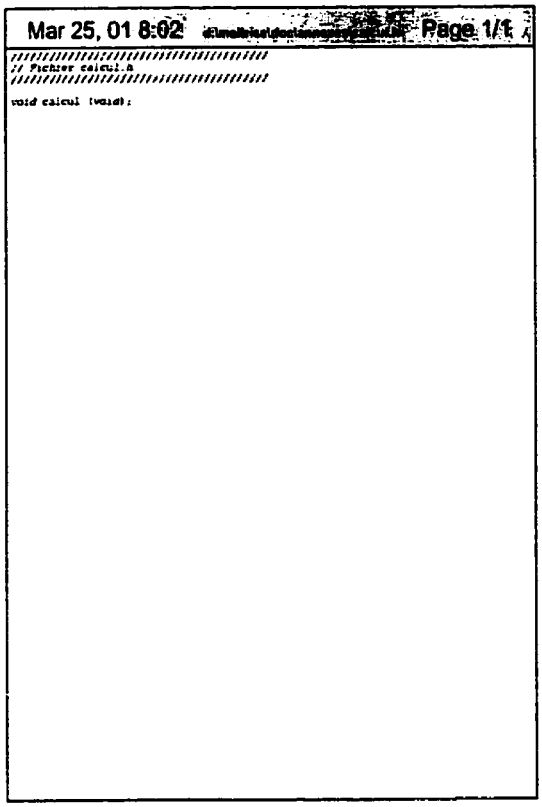
Mar 25, 01 8:59 d:\maitrise\declanneau\obstr.c Page 2/2

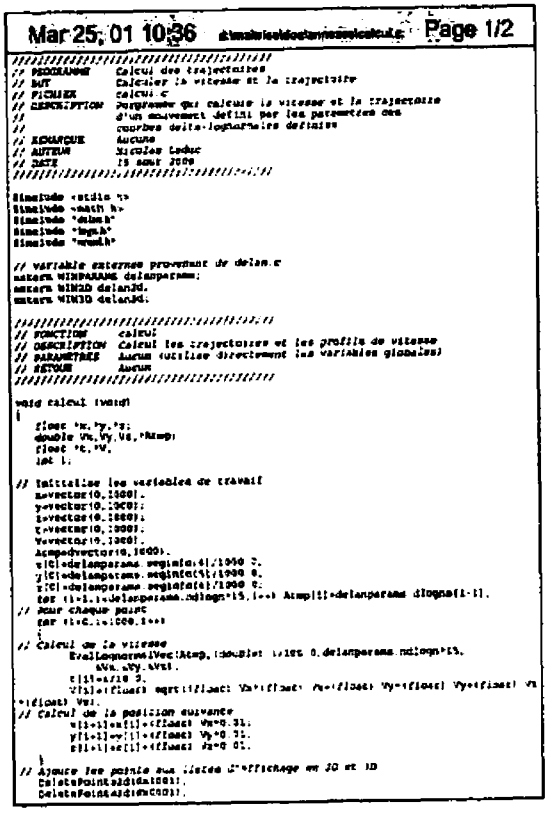
```
   free_dvector(x,1,Nbpt);
   free_dvector(y,1,Nbpt);
   free_dvector(z,1,Nbpt);
   free_dvector(DX,1,Nbpt);
   free_dvector(DY,1,Nbpt);
   free_dvector(DZ,1,Nbpt);
   free_dvector(v,1,Nbpt);
   free_dvector(V,1,Nbpt);
   free_dvector(VF,1,Nbpt);
   return (0);
}
```

Annexe 6

Visualisation de trajectoires

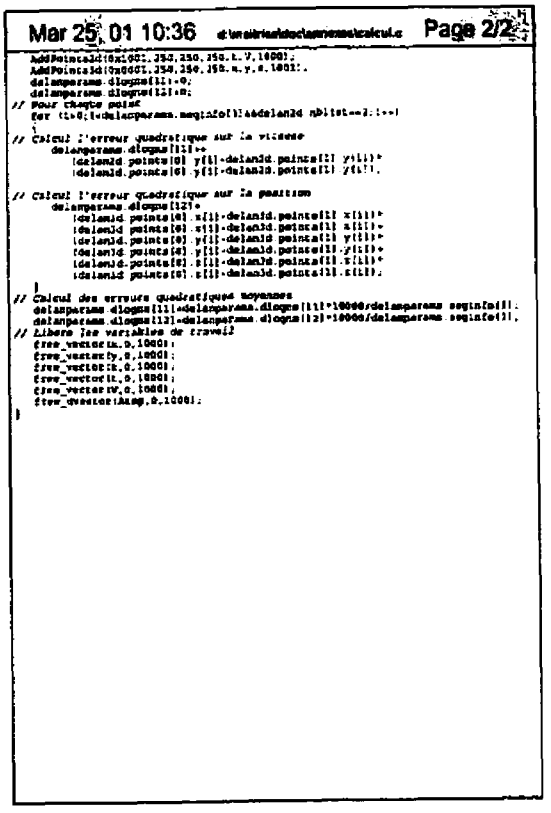
Cette annexe présente le programme qui nous a permis de terminer l'extraction des paramètres pour les mouvements simples et de visualiser la reconstruction des trajectoires et des profils de vitesse. Celui-ci a été programmé en utilisant les routines Win32 pour l'interface ainsi que la librairie OpenGL pour l'affichage des courbes en trois dimensions. Nous avons utilisé le compilateur GNU (gcc) pour compiler le programme. L'utilisation de ce dernier nécessite l'installation de DJGPP (programmes GNU pour DOS) et de RSXNTDJ (outils de compilation Win32). Les sources sont disponibles par courriel sur demande à hexabit@step.polymtl.ca.


Mar 25, 01 8:02
Page 1/1
// Fichier calcul.h
void calcul (void);



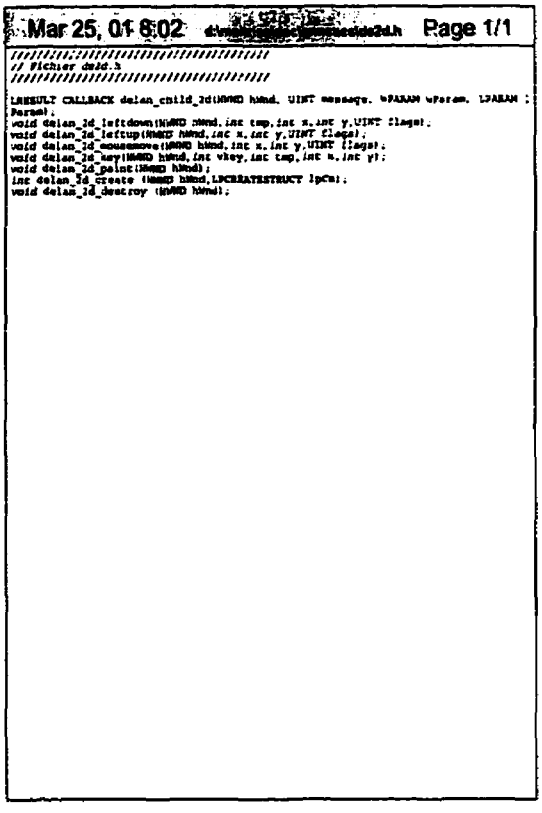
Mar 25, 01 10:36 d:\mabrisebdoctorat\prog\calcul.c Page 1/2

```
////////////////////////////////////////////////////
// PROGRAMME     Calcul des trajectoires
// BUT           Calculer la vitesse et la trajectoire
// FICHIER       calcul.c
// DESCRIPTION   Programme qui calcule la vitesse et la trajectoire
//               d'un mouvement défini par les parametres des
//               courbes delta-lognormales definies
// REMARQUE      Aucune
// AUTEUR        Nicolas Leduc
// DATE          15 mai 2000
////////////////////////////////////////////////////

#include <stdio.h>
#include <math.h>
#include "delan.h"
#include "bgm.h"
#include "vrml.h"

// variable externes provenant de delan.c
extern WINPARAMS delanparams;
extern WIN2D delan2d.
extern WIN3D delan3d.

////////////////////////////////////////////////////
// FONCTION      calcul
// DESCRIPTION   Calcul les trajectoires et les profils de vitesse
// PARAMETRES    Aucun (utilise directement les variables globales)
// RETOUR        Aucun
////////////////////////////////////////////////////

void calcul (void)
{
   float *x,*y,*s;
   double Vx,Vy,Vs,*Atmp;
   float *t,*V,
   int i,

// Initialise les variables de travail
   x=vector(0,1000).
   y=vector(0,1000);
   t=vector(0,1000);
   t=vector(0,1000).
   V=vector(0,1000).
   Atmp=dvector(0,1000).
   x[0]=delanparams.seginfo[0].x/1000 0.
   y[0]=delanparams.seginfo[0].y/1000 0.
   z[0]=delanparams.seginfo[0].z/1000 0.
   for (i=1.i<delanparams.nblogn*15,i++) Atmp[i]=delanparams dlogna[i-1].
// Pour chaque point
   for (t=0.t<1000.t++)
   {
// Calcul de la vitesse
      EvalLognormalVec(Atmp,(double) t/100 0.delanparams.nblogn*15.
            &Vx.&Vy.&Vs).
      t[i]=s/10 3.
      V[i]=(float) sqrt((float) Vx*(float) Vx+(float) Vy*(float) Vy+(float) Vz
*(float) Vz).
// Calcul de la position suivante
      x[i+1]=x[i]+(float) Vx*0.01.
      y[i+1]=y[i]+(float) Vy*0 01.
      z[i+1]=z[i]+(float) Vz*0 01.
   }
// Ajoute les points aux listes d'affichage en 2D et 3D
   DeletePoint2d(delan2d);
   DeletePoint3d(delan3d);
```



Mar 25, 01 10:36 d:\mabrisebdoctorat\prog\calcul.c Page 2/2

```
   AddPoint2d(0x1002,250,250,250,t,V,1000);
   AddPoint3d(0x0007,250,250,250,x,y,z,1002);
   delanparams.dlogna[1]=0;
   delanparams.dlogna[2]=0;
// Pour chaque point
   for (t=0;t<delanparams.seginfo[0].nbPlot-2;t++)
   {
// Calcul l'erreur quadratique sur la vitesse
      delanparams.dlogna[1]+=
         (delan2d.points[0].y[t]-delan2d.points[2].y[t])*
         (delan2d.points[0].y[t]-delan2d.points[2].y[t]).

// Calcul l'erreur quadratique sur la position
      delanparams.dlogna[2]+
         (delan2d.points[0].x[t]-delan2d.points[2].x[t])*
         (delan2d.points[0].x[t]-delan2d.points[2].x[t])+
         (delan2d.points[0].y[t]-delan2d.points[2].y[t])*
         (delan2d.points[0].y[t]-delan2d.points[2].y[t])+
         (delan2d.points[0].z[t]-delan2d.points[2].z[t])*
         (delan2d.points[0].z[t]-delan2d.points[2].z[t]).
   }
// Calcul des erreurs quadratiques moyennes
   delanparams.dlogna[1]=delanparams.dlogna[1]*10000/delanparams.seginfo[0].
   delanparams.dlogna[2]=delanparams.dlogna[2]*10000/delanparams.seginfo[0].
// Libere les variables de travail
   free_vector(x,0,1000);
   free_vector(y,0,1000);
   free_vector(z,0,1000);
   free_vector(t,0,1000);
   free_vector(V,0,1000);
   free_dvector(Atmp,0,1000);
}
```



Mar 25, 01 8:02 [illegible] Page 1/1

```
//////////////////////////////////////
// Fichier delc.h
//////////////////////////////////////

LRESULT CALLBACK delan_child_2d(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
void delan_2d_leftdown(HWND hWnd,int tmp,int x,int y,UINT flags);
void delan_2d_leftup(HWND hWnd,int x,int y,UINT flags);
void delan_2d_mousemove(HWND hWnd,int x,int y,UINT flags);
void delan_2d_key(HWND hWnd,int vkey,int tmp,int x,int y);
void delan_2d_paint(HWND hWnd);
int delan_2d_create (HWND hWnd,LPCREATESTRUCT lpCs);
void delan_2d_destroy (HWND hWnd);
```

```
//////////////////////////////////////////
// PROGRAMME    Affichage en 2D
// BUT          Gerer la fenetre d'affichage en 2D
// FICHIER      de3d.c
// DESCRIPTION  Permet de gerer l'affichage des profils de vitesse
//              dans la fenetre 2D
// REMARQUE     Aucune
// AUTEUR       Nicolas Leduc
// DATE         15 aout 2001
//////////////////////////////////////////

#include <stdio.h>
#include <windows.h>
#include <windowsx.h>
#include <math.h>
#include <malloc.h>
#include "delan.h"
#include "de3d.h"

// Variables globales de delan.c
extern HINSTANCE delan3d;
extern WINPARAMS delanparams;

//////////////////////////////////////////
// FONCTION     delan_child_3d
// DESCRIPTION  handler des messages
//////////////////////////////////////////

LRESULT CALLBACK delan_child_3d(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_PAINT   : return HANDLE_WM_PAINT(hWnd,wParam,lParam,delan_3d_paint);
        case WM_DESTROY : return HANDLE_WM_DESTROY(hWnd,wParam,lParam,delan_3d_destroy);
        case WM_LBUTTONDOWN : return HANDLE_WM_LBUTTONDOWN(hWnd,wParam,lParam,delan_3d_leftdown);
        case WM_LBUTTONUP : return HANDLE_WM_LBUTTONUP(hWnd,wParam,lParam,delan_3d_leftup);
        case WM_MOUSEMOVE : return HANDLE_WM_MOUSEMOVE(hWnd,wParam,lParam,delan_3d_mousemove);
        case WM_CREATE    : return HANDLE_WM_CREATE(hWnd,wParam,lParam,delan_3d_create);
        case WM_KEYDOWN   : return HANDLE_WM_KEYDOWN(hWnd,wParam,lParam,delan_3d_key);
    }
    return DefWindowProc(hWnd,message,wParam,lParam);
}

//////////////////////////////////////////
// FONCTION     delan_3d_leftdown
// DESCRIPTION  Gere la pression du bouton de gauche de la souris
// PARAMETRES   hWnd: handler de la fenetre 2D
//              tmp: inutilise
//              x,y: position de la souris
//              flags: drapeaux de statut
// RETOUR       Aucun
//////////////////////////////////////////

void delan_3d_leftdown(HWND hWnd,int tmp,int x,int y,UINT flags)
{
    SetFocus(hWnd);
// Si l'usager presse SHIFT en meme temps que le bouton, on en
```



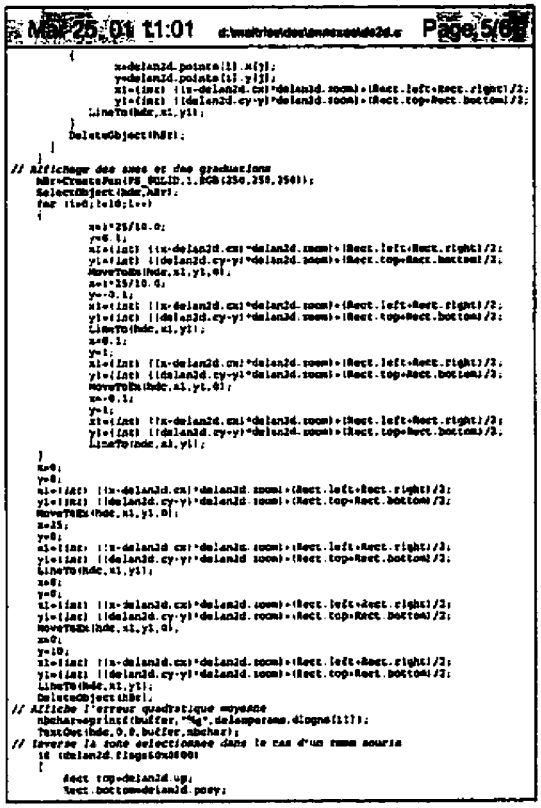
```
// prend note
    if (GetKeyState(VK_SHIFT)&128. delan3d.flags[=0x0400;
    else delan3d.flags[=0x0400;
// On sauvegarde la position courante de la souris
    delan3d.psavex=x;
    delan3d.psavey=y;
    delan3d.up=y;
    delan3d.left=x;
}

//////////////////////////////////////////
// FONCTION     delan_3d_leftup
// DESCRIPTION  Le bouton gauche de la souris a ete relache
// PARAMETRES   hWnd: handler de la fenetre 3d
//              x,y: position de la souris
//              flags: drapeaux indiquant le statut de la souris
// RETOUR       Aucun
//////////////////////////////////////////

void delan_3d_leftup(HWND hWnd,int x,int y,UINT flags)
{
    RECT Rect;
    HDC hDC;
    float vzoom,hzoom;

    GetClientRect(hWnd,&Rect);
// Si on faisait du dragdrop, on le desactive
    if (delan3d.flags&0x0400)
    {
        delan3d.flags^=0x0400;
    }
// Si on etait en mode zoom
    if (delan3d.flags&0x0800)
    {
// desactivation du mode
        delan3d.flags^=0x0800;
        hDC=GetDC(hWnd);
// Recentrer l'image d'apres la selection
        delan3d.cx+=((delan3d.left+x)/2-(Rect.left+Rect.right)/2)/delan3d.zoom;
        delan3d.cy+=((Rect.bottom+Rect.top)/2-(delan3d.up+y)/2)/delan3d.zoom;
// Calcul du facteur de zoom
        vzoom=(float) 1.5;
        hzoom=(float) 1.5;
        if (delan3d.up-y!=0) vzoom=(float) fabs((Rect.bottom-Rect.top)*((float)delan3d.up-y));
        if (delan3d.left-x!=0) hzoom=(float) fabs((Rect.right-Rect.left)*((float)delan3d.left-x));
// On efface la selection en couleurs inversees
        Rect.top=delan3d.up;
        Rect.bottom=delan3d.psavy;
        Rect.left=delan3d.left;
        Rect.right=delan3d.psavx;
        InvertRect(hDC,&Rect);
        if (vzoom>hzoom) vzoom=hzoom;
        delan3d.zoom*=vzoom;
        if (delan3d.zoom>6.001) delan3d.zoom=(float) 20.
        ReleaseDC(hWnd,hDC);
// Afficher la nouvelle image
        InvalidateRect(hWnd,0,1);
    }
}

//////////////////////////////////////////
// FONCTION     delan_3d_mousemove
// DESCRIPTION  Actions lorsque la souris se deplace
```



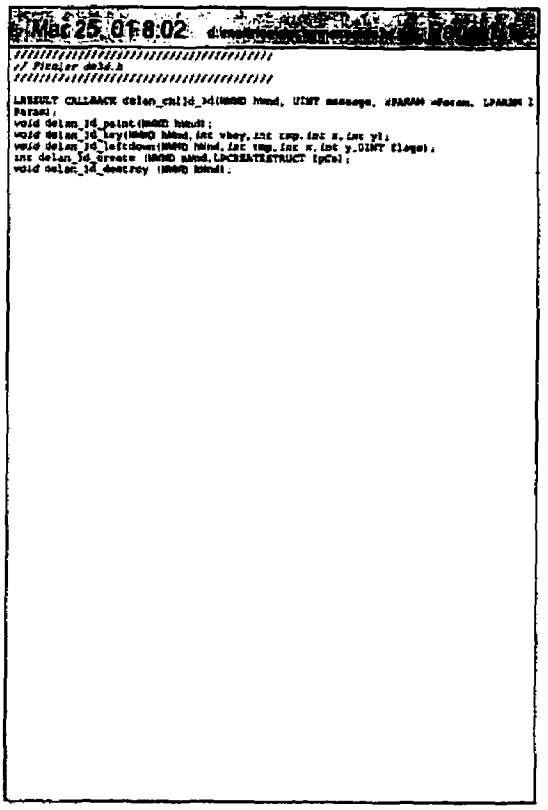
```
// PARAMETRES   hWnd: handler de la fenetre 3D
//              x,y: position de la souris
//              flags: drapeaux indiquant le statut de la souris
// RETOUR       Aucun
//////////////////////////////////////////

void delan_3d_mousemove(HWND hWnd,int x,int y,UINT flags)
{
    RECT Rect;
    HDC hDC;

// Si on fait du drag&drop
    if !delan3d.flags&0x0400)
    {
// On deplace le centre de l'image et on reaffiche
        delan3d.cx+=(delan3d.psavx-x)/delan3d.zoom;
        delan3d.cy+=(y-delan3d.psavy)/delan3d.zoom;
        delan3d.psavx=x;
        delan3d.psavy=y;
        InvalidateRect(hWnd,0,1);
    }
// Si on fait du zoom
    if (delan3d.flags&0x0800)
    {
// On efface la selection courante
        hDC=GetDC(hWnd);
        Rect.top=delan3d.up;
        Rect.bottom=delan3d.psavy;
        Rect.left=delan3d.left;
        Rect.right=delan3d.psavx;
        InvertRect(hDC,&Rect);
// On note la nouvelle selection
        delan3d.psavx=x;
        delan3d.psavy=y;
        Rect.bottom=delan3d.psavy;
        Rect.right=delan3d.psavx;
// On reaffiche
        InvertRect(hDC,&Rect);
        ReleaseDC(hWnd,hDC);
    }
}

//////////////////////////////////////////
// FONCTION     delan_3d_key
// DESCRIPTION  Traitements des touches du clavier pour la fenetre 2D
// PARAMETRES   hWnd: handler de la fenetre 2D
//              vkey: code la touche pressee
//              tmp,x,y: inutilise
// RETOUR       Aucun
//////////////////////////////////////////

void delan_3d_key(HWND hWnd,int vkey,int tmp,int x,int y)
{
// Zoom +
    if (vkey==VK_ADD)
    {
        delan3d.zoom*=(float) 1.5;
    }
// Deplacement vers le haut
    if (vkey==VK_UP)
    {
        delan3d.cy+=(float) 5.0/delan3d.zoom;
    }
// Deplacement vers le bas
    if (vkey==VK_DOWN)
```



```
    {
        delan3d.cy-=(float) 5.0/delan3d.zoom;
    }
// Deplacement vers la gauche
    if (vkey==VK_LEFT)
    {
        delan3d.cx+=(float) 5.0/delan3d.zoom;
    }
// Deplacement vers la droite
    if (vkey==VK_RIGHT)
    {
        delan3d.cx-=(float) 5.0/delan3d.zoom;
    }
// Zoom -
    if (vkey==VK_SUBTRACT)
    {
        delan3d.zoom/=(float) 1.5;
    }
// Ramene l'image a la position initiale
    if (vkey==VK_CLEAR)
    {
        delan3d.zoom=(float)20.0;
        delan3d.cx=(float)0.0;
        delan3d.cy=(float)0.0;
    }
    InvalidateRect(hWnd,0,1);
}

//////////////////////////////////////////
// FONCTION     delan_3d_paint
// DESCRIPTION  Rafraichissement de la fenetre 3D
// PARAMETRES   hWnd: handler de la fenetre 3D
// RETOUR       Aucun
//////////////////////////////////////////

void delan_3d_paint(HWND hWnd)
{
    PAINTSTRUCT ps;
    HDC hdc;
    HBRUSH hBr;
    RECT Rect;
    float x,y;
    int xi,yi,i,j,nbchar;
    char buffer[100];

    GetClientRect(hWnd,&Rect);
    hdc=BeginPaint(hWnd,&ps);
// Pour chacune des listes de points
    for (i=0;i<delan3d.nblist;i++)
    {
// S'il faut afficher les points
        if (delan3d.flags&delan3d.points[i].onflags&0x7C006A
            delan3d.flags&delan3d.points[i].onflags&0x01FF)
        {
// Choisi la couleur a utiliser
            hBr=CreateHatchBrush(HS_SOLID,RGB(delan3d.points[i].R,delan3d.points[i].G
,delan3d.points[i].B));
            SelectObject(hdc,hBr);
// Affiche les points
            xi=delan3d.points[i].x[0];
            yi=delan3d.points[i].y[0];
            xi=(int) ((x-delan3d.cx)*delan3d.zoom)+(Rect.left+Rect.right)/2;
            yi=(int) ((delan3d.cy-y)*delan3d.zoom)+(Rect.top+Rect.bottom)/2.
            MoveToEx(hdc,xi,yi,0);
            for (j=1;j<delan3d.points[i].nbpts;j++)
```



```
Mar 25 01 11:01                                                    Page 5/6

        {
                x=delan2d.points[i].x[y];
                y=delan2d.points[i].y[y];
                xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
                yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
            LineTo(hdc,xi,yi);
        }
        DeleteObject(hBr);
    }
// Affichage des axes et des graduations
    hBr=CreatePen(PS_SOLID,1,RGB(250,250,250));
    SelectObject(hdc,hBr);
    for (i=0;i<10;i++)
    {
            x=i*25/10.0;
            y=0.1;
            xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
            yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
            MoveToEx(hdc,xi,yi,0);
            x=i*25/10.0;
            y=-0.1;
            xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
            yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
            LineTo(hdc,xi,yi);
            x=0.1;
            y=i;
            xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
            yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
            MoveToEx(hdc,xi,yi,0);
            x=-0.1;
            y=i;
            xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
            yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
            LineTo(hdc,xi,yi);
    }
    x=0;
    y=0;
    xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
    yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
    MoveToEx(hdc,xi,yi,0);
    x=25;
    y=0;
    xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
    yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
    LineTo(hdc,xi,yi);
    x=0;
    y=0;
    xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
    yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
    MoveToEx(hdc,xi,yi,0);
    x=0;
    y=10;
    xi=(int) ((x-delan2d.cx)*delan2d.zoom)+(Rect.left+Rect.right)/2;
    yi=(int) ((delan2d.cy-y)*delan2d.zoom)+(Rect.top+Rect.bottom)/2;
    LineTo(hdc,xi,yi);
    DeleteObject(hBr);
// Affiche l'erreur quadratique moyenne
    nbchar=sprintf(buffer,"%g",delanparams.diagnol[1]);
    TextOut(hdc,0,0,buffer,nbchar);
// Ecrire la zone selectionnee dans le cas d'un zoom souris
    if (delan2d.flags&0x0800)
    {
        Rect.top=delan2d.up;
        Rect.bottom=delan2d.posy;
        Rect.left=delan2d.left;
        Rect.right=delan2d.posx;
        InvertRect(hdc,&Rect);
    }
    EndPaint(hwnd,&ps);
}

////////////////////////////////////////
// FONCTION     delan_2d_create
// DESCRIPTION  Creation de la fenetre 2D
// PARAMETRES   hWnd: handler de la fenetre 2D
//              lpCs: structure renseignant sur la creation
// RETOUR       int: 1 si tout est OK, 0 sinon
////////////////////////////////////////

int delan_2d_create (HWND hWnd,LPCREATESTRUCT lpCs)
{
// Initialisation des structures
    delan2d.hWnd=hWnd;
    delan2d.nblist=0;
    delan2d.points=0;
    delan2d.cx=0;
    delan2d.cy=0;
    delan2d.zoom=10;
    delan2d.flags=0;
    delan2d.posX=0;
    delan2d.posy=0;
    delan2d.up=0;
    delan2d.left=0;
    return 1;
}

////////////////////////////////////////
// FONCTION     delan_2d_destroy
// DESCRIPTION  Destructeur de la fenetre 2D
// PARAMETRES   hWnd: handler de la fenetre 2D
// RETOUR       Aucun
////////////////////////////////////////

void delan_2d_destroy (HWND hWnd)
{
    int i;

// Liberation de la memoire occupee par les structures de travail
    for (i=0;i<delan2d.nblist;i++)
    {
        free(delan2d.points[i].x);
        free(delan2d.points[i].y);
    }
    free(delan2d.points);
    delan2d.nblist=0;
}
```


Mar 25 01:8:02
// FileJer delA.h
LRESULT CALLBACK delan_child_jd(HWND hwnd, UINT message, WPARAM wParam, LPARAM l
Param);
void delan_jd_paint(HWND hWnd);
void delan_jd_key(HWND hWnd,int vkey,int cmp,int x,int y);
void delan_jd_leftdown(HWND hWnd,int tmp,int x,int y,UINT flags);
int delan_jd_create (HWND aWnd,LPCREATESTRUCT lpCs);
void delan_jd_destroy (HWND hWnd);

Mar 25, 01 11:43 Page 1/6

```
//////////////////////////////////////////
// PROGRAMME    Gestion de la fenetre 3D
// BUT          Faire la gestion de l'affichage 3D
// FICHIER      deld.c
// DESCRIPTION  Cree la fenetre qui permet d'afficher les
//              trajectoires en 3D
// REMARQUE     Utilisation de OpenGL
// AUTEUR       Nicolas Ledun
// DATE         15 aout 2001
//////////////////////////////////////////

#include <stdio.h>
#include <windows.h>
#include <windowsx.h>
#include <math.h>
#include <malloc.h>
#include <GL\gl.h>
#include <GL\glu.h>
#include "dlan.h"
#include "deld.h"
#include "resold"

// Variable globales de delan.c
extern HINSTANCE delanInd;
extern WINPARAMS delanparams;

//////////////////////////////////////////
// FONCTION     delan_child_3d
// DESCRIPTION  Handler des messages de la fenetre 3D
//////////////////////////////////////////

LRESULT CALLBACK delan_child_3d(HWND hWnd, UINT message, WPARAM wParam, LPARAM l
Param)
{
    switch (message)
    {
        case WM_PAINT   : return HANDLE_WM_PAINT(hWnd,wParam,lParam,delan_3d_pai
nt);
        case WM_DESTROY : return HANDLE_WM_DESTROY(hWnd,wParam,lParam,delan_3d
_destroy);
        case WM_CREATE  : return HANDLE_WM_CREATE(hWnd,wParam,lParam,delan_3d_c
reate);
        case WM_KEYDOWN       return HANDLE_WM_KEYDOWN(hWnd,wParam,lParam,delan
_3d_key);
        case WM_LBUTTONDOWN : return HANDLE_WM_LBUTTONDOWN(hWnd,wParam,lParam
,delan_3d_leftdown);
    }
    return DefWindowProc(hWnd,message,wParam,lParam);
}

//////////////////////////////////////////
// FONCTION     delan_3d_paint
// DESCRIPTION  Affichage des courbes en 3D
// PARAMETRES   hWnd: handler de la fenetre 3D
// RETOUR       Aucun
//////////////////////////////////////////

void delan_3d_paint(HWND hWnd)
{
    PAINTSTRUCT ps;
    HDC hdc;
     int i,j,nbchar;
    char buffer[100];

    hdc=BeginPaint(hWnd,&ps);
```

Mar 25, 01 11:43 Page 2/6

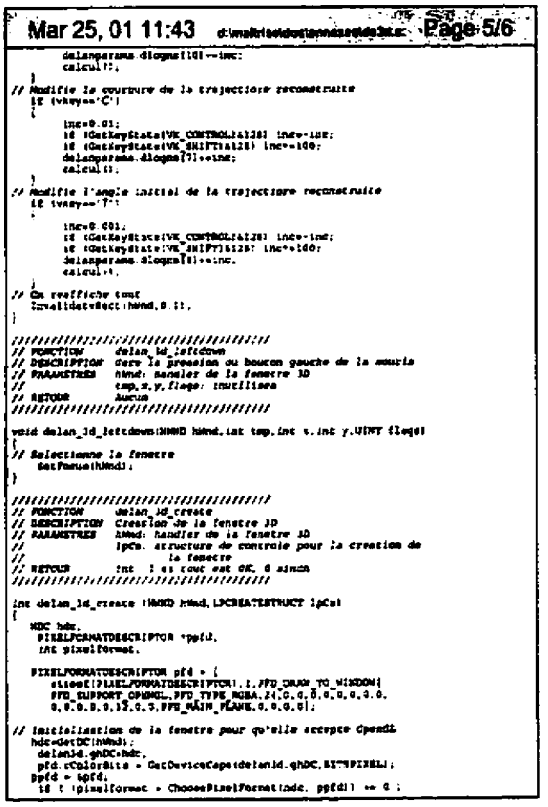
```
// On remet les matrices GL a zero
   glClear(GL_COLOR_BUFFER_BIT);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
// On place le point de vue
    glLookAt(delanId.EyeX,delanId.EyeY,delanId.EyeZ,
         delanId.EyeX+cos(delanId.Roull)*sin(delanId.Theta),
         delanId.EyeY+sin(delanId.Roull),
         delanId.EyeZ+cos(delanId.Roull)*cos(delanId.Theta),
         cos(delanId.Roull-PI/2)*sin(delanId.Theta),
         sin(delanId.Roull+PI/2),
         cos(delanId.Roull-PI/2)*cos(delanId.Theta));
// On choisi le type de vue
    glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-0.2,0.2,-0.2,0.2,1.28,5);
// Pour toutes les listes
    for (i=0;i<delanId.nblist;i++)
    {
// Si on doit afficher la liste
        if (delanId.flagsdelanId.points[i].dmflag)
        {
// On affiche chaque point de la liste courante
            glBegin(GL_LINE_STRIP);
            glColor3d(delanId.points[i].R,delanId.points[i].G,delanId.points[i]
.B);
            for (j=0;j<delanId.points[i].nbpts;j++)
            {
                glVertex3d(delanId.points[i].x[j],delanId.points[i].y[j],delanId
.points[i].z[j]);
            }
            glEnd();
        }
    }
// Affichage des axes
    glBegin(GL_LINES);
      glColor3d(1.0,0.0,0.0);
      glVertex3d(0,0,0);
      glVertex3d(10,0,0);
      glColor3d(0.0,1.0,0.0);
      glVertex3d(0,0,0);
      glVertex3d(0,10,0);
      glColor3d(0.0,0.0,1.0);
      glVertex3d(0,0,0);
      glVertex3d(0,0,10);
// Affichage des graduations
      for (i=0;i<15;i++)
      {
        glColor3d(1.0,0.0,0.0);
        glVertex3d(i/10.0,-0.01,0.0);
        glVertex3d(i/10.0,0.01,0.0);
        glVertex3d(i/10.0,0.0,-0.01);
        glVertex3d(i/10.0,0.0,0.01);
        glColor3d(0.0,1.0,0.0);
        glVertex3d(-0.01,i/10.0,0.0);
        glVertex3d(0.01,i/10.0,0.0);
        glVertex3d(0.0,i/10.0,-0.01);
        glVertex3d(0.0,i/10.0,0.01);
        glColor3d(0.0,0.0,1.0);
        glVertex3d(-0.01,0.0,i/10.0);
        glVertex3d(0.01,0.0,i/10.0);
        glVertex3d(0.0,-0.01,i/10.0);
        glVertex3d(0.0,0.01,i/10.0);
      }
    glEnd();
```

Mar 25, 01 11:43 Page 3/6

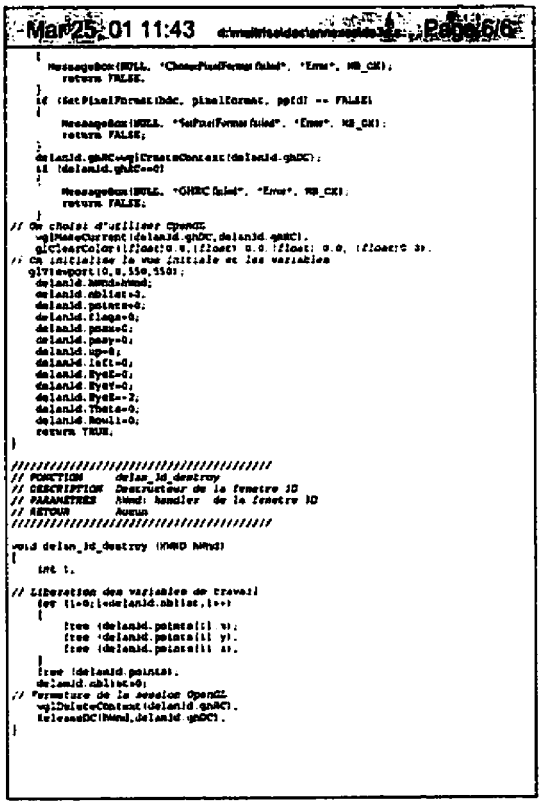
```
// on ordonne un reaffichage
    glFlush();
    nbchar=sprintf(buffer,"%g",delanparams.dlogne[12]);
    TextOut(hdc,0,0,buffer,nbchar);
    EndPaint(hWnd,&ps);
}

//////////////////////////////////////////
// FONCTION     delan_3d_key
// DESCRIPTION  Gestion des touches du clavier
// PARAMETRES   hWnd: handler de la fenetre 3D
//              vkey: code de la touche appuyee
//              tmp,x,y: inutilise
// RETOUR       Aucun
//////////////////////////////////////////

void delan_3d_key(HWND hWnd,int vkey,int tmp,int x,int y)
{
    double inc;

// Monte
    if (vkey=='I')
    {
        delanId.EyeX+=0.1*cos(delanId.Roull+PI/2)*sin(delanId.Theta);
        delanId.EyeY+=0.1*sin(delanId.Roull+PI/2);
        delanId.EyeZ+=0.1*cos(delanId.Roull+PI/2)*cos(delanId.Theta);
    }
// Descend
    if (vkey=='J')
        delanId.EyeX-=0.1*cos(delanId.Roull+PI/2)*sin(delanId.Theta);
        delanId.EyeY-=0.1*sin(delanId.Roull+PI/2);
        delanId.EyeZ-=0.1*cos(delanId.Roull+PI/2)*cos(delanId.Theta);
    }
// Gauche
    if (vkey=='S')
        delanId.EyeX-=0.1*cos(0)*sin(delanId.Theta-PI/2);
        delanId.EyeY-=0.1*sin(0);
        delanId.EyeZ-=0.1*cos(0)*cos(delanId.Theta-PI/2);
    }
// Droite
    if (vkey=='F')
        delanId.EyeX+=0.1*cos(0)*sin(delanId.Theta-PI/2);
        delanId.EyeY+=0.1*sin(0);
        delanId.EyeZ+=0.1*cos(0)*cos(delanId.Theta-PI/2);
    }
// Avance
    if (vkey==VK_INSERT)
    {
        delanId.EyeX+=0.1*cos(delanId.Roull)*sin(delanId.Theta);
        delanId.EyeY+=0.1*sin(delanId.Roull);
        delanId.EyeZ+=0.1*cos(delanId.Roull)*cos(delanId.Theta);
    }
// Recule
    if (vkey==VK_DELETE)
    {
        delanId.EyeX-=0.1*cos(delanId.Roull)*sin(delanId.Theta);
        delanId.EyeY-=0.1*sin(delanId.Roull);
        delanId.EyeZ-=0.1*cos(delanId.Roull)*cos(delanId.Theta);
    }
// Tourne a gauche
    if (vkey==VK_LEFT)
        delanId.Theta+=PI/64;
        if(delanId.Theta>3*PI-PI/128) delanId.Theta=0;
    }
// Tourne a droite
```

Mar 25, 01 11:43 Page 4/6

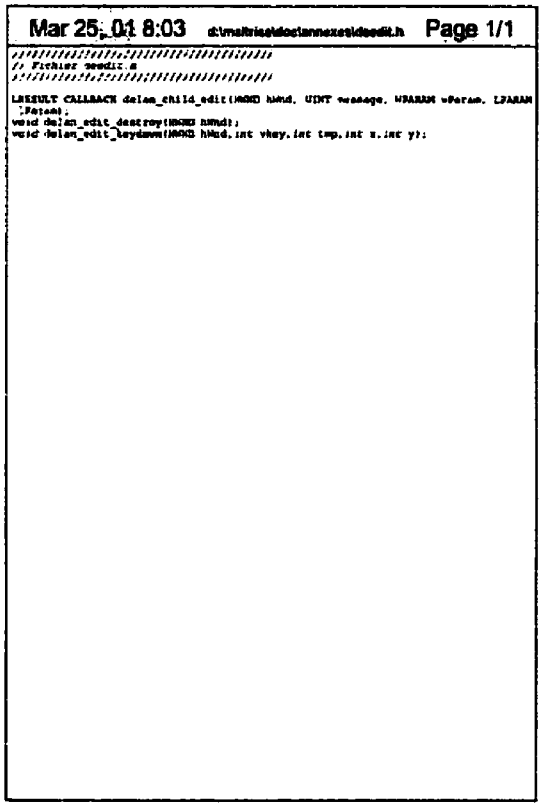
```
    if (vkey==VK_RIGHT)
        delanId.Theta-=PI/64;
        if(delanId.Theta<-3*PI+PI/128) delanId.Theta=0;
    }
// Tourne vers le Haut
    if (vkey==VK_UP)
        delanId.Roull+=PI/64;
        if(delanId.Roull>2*PI-PI/128) delanId.Roull=0;
    }
// Tourne vers le bas
    if (vkey==VK_DOWN)
        delanId.Roull-=PI/64;
        if(delanId.Roull<-2*PI+PI/128) delanId.Roull=0;
    }
// Retour a la position initiale
    if (vkey==VK_CLEAR)
    {
        delanId.Theta=0;
        delanId.Roull=0;
        delanId.EyeX=0;
        delanId.EyeY=0;
        delanId.EyeZ=-2;
    }
// Modifie la position initiale en X de la trajectoire reconstruite
    if (vkey=='X')
    {
        inc=1;
        if (GetKeyState(VK_CONTROL)&128) inc=-inc;
        if (GetKeyState(VK_SHIFT)&128) inc*=100;
        delanparams.seginfo[4]+=inc;
        calcul();
    }
// Modifie la position initiale en Y de la trajectoire reconstruite
    if (vkey=='Y')
    {
        inc=1;
        if (GetKeyState(VK_CONTROL)&128) inc=-inc;
        if (GetKeyState(VK_SHIFT)&128) inc*=100;
        delanparams.seginfo[5]+=inc;
        calcul();
    }
// Modifie la position initiale en Z de la trajectoire reconstruite
    if (vkey=='Z')
    {
        inc=1;
        if (GetKeyState(VK_CONTROL)&128) inc=-inc;
        if (GetKeyState(VK_SHIFT)&128) inc*=100;
        delanparams.seginfo[6]+=inc;
        calcul();
    }
// Modifie la rotation autour de X de la trajectoire reconstruite
    if (vkey=='R')
    {
        inc=0.001;
        if (GetKeyState(VK_CONTROL)&128) inc=-inc;
        if (GetKeyState(VK_SHIFT)&128) inc*=100;
        delanparams.dlogne[9]+=inc;
        calcul();
    }
// Modifie la rotation autour de X de la trajectoire reconstruite
    if (vkey=='E')
    {
        inc=0.001;
        if (GetKeyState(VK_CONTROL)&128) inc=-inc;
        if (GetKeyState(VK_SHIFT)&128) inc*=100;
```

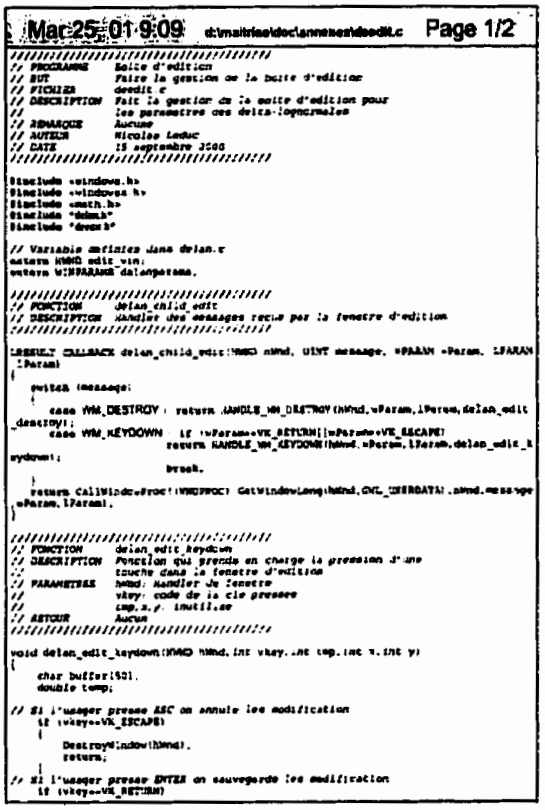


```
Mar 25, 01 11:43   d:\multi\delan\...   Page 5/6

        delanparams.dlogma[10]+=inc;
        calcul();
    }
// Modifie la courbure de la trajectoire reconstruite
    if (vkey=='C')
    {
        inc=0.01;
        if (GetKeyState(VK_CONTROL)&128) inc=-inc;
        if (GetKeyState(VK_SHIFT)&128) inc=inc*100;
        delanparams.dlogma[7]+=inc;
        calcul();
    }
// Modifie l'angle initial de la trajectoire reconstruite
    if (vkey=='T')
    {
        inc=0.001;
        if (GetKeyState(VK_CONTROL)&128) inc=-inc;
        if (GetKeyState(VK_SHIFT)&128) inc=inc*100;
        delanparams.dlogma[8]+=inc;
        calcul();
    }
// On reaffiche tout
    InvalidateRect(hWnd,0,1);
}

////////////////////////////////////////
// FONCTION     delan_3d_leftdown
// DESCRIPTION  Gere la pression du bouton gauche de la souris
// PARAMETRES   hWnd: handler de la fenetre 3D
//              tmp,x,y,flags: inutilises
// RETOUR       Aucun
////////////////////////////////////////

void delan_3d_leftdown(HWND hWnd,int tmp,int x,int y,UINT flags)
{
// Selectionne la fenetre
    SetFocus(hWnd);
}

////////////////////////////////////////
// FONCTION     delan_3D_create
// DESCRIPTION  Creation de la fenetre 3D
// PARAMETRES   hWnd: handler de la fenetre 3D
//              lpCs: structure de controle pour la creation de
//                    la fenetre
// RETOUR       int  : 0 si tout est OK, 0 sinon
////////////////////////////////////////

int delan_3d_create (HWND hWnd,LPCREATESTRUCT lpCs)
{
    HDC hdc;
    PIXELFORMATDESCRIPTOR *ppfd;
    int pixelformat;

    PIXELFORMATDESCRIPTOR pfd = {
        sizeof(PIXELFORMATDESCRIPTOR),1,PFD_DRAW_TO_WINDOW|
        PFD_SUPPORT_OPENGL,PFD_TYPE_RGBA,24,0,0,0,0,0,0,0,
        0,0,0,0,0,32,0,0,PFD_MAIN_PLANE,0,0,0,0};

// Initialisation de la fenetre pour qu'elle accepte OpenGL
    hdc=GetDC(hWnd);
    delan3d.ghDC=hdc;
    pfd.cColorBits = GetDeviceCaps(delan3d.ghDC,BITSPIXEL);
    ppfd = &pfd;
    if ( !(pixelformat = ChoosePixelFormat(hdc, ppfd)) == 0 )
```



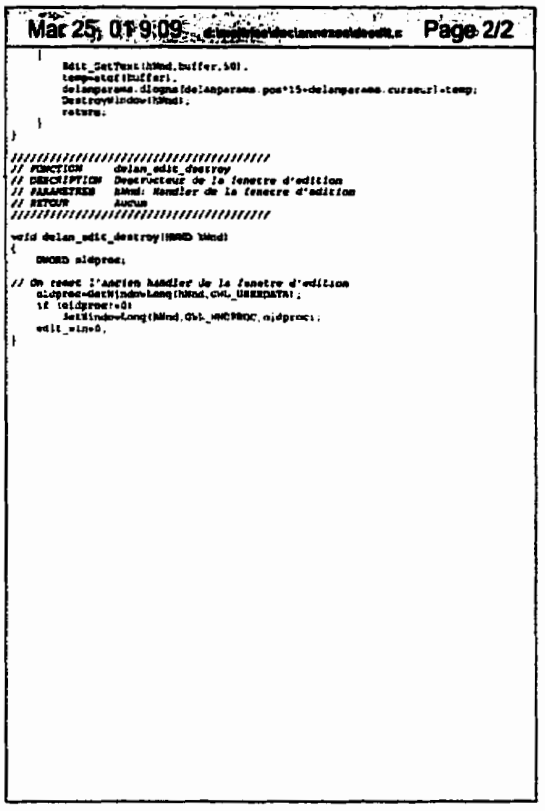
```
Mar 25, 01 11:43   d:\multi\delan\...   Page 6/6

    {
        MessageBox(NULL, "ChoosePixelFormat failed", "Error", MB_OK);
        return FALSE;
    }
    if (SetPixelFormat(hdc, pixelformat, ppfd) == FALSE)
    {
        MessageBox(NULL, "SetPixelFormat failed", "Error", MB_OK);
        return FALSE;
    }
    delan3d.ghRC=wglCreateContext(delan3d.ghDC);
    if (delan3d.ghRC==0)
    {
        MessageBox(NULL, "GHRC failed", "Error", MB_OK);
        return FALSE;
    }
// On choisit d'utiliser OpenGL
    wglMakeCurrent(delan3d.ghDC,delan3d.ghRC);
    glClearColor((float)0.0,(float) 0.0 ,(float) 0.0, (float)0 );
// On initialise la vue initiale et les variables
    glViewport(0,0,550,550);
    delan3d.hWnd=hWnd;
    delan3d.nblist=3;
    delan3d.points=0;
    delan3d.flags=0;
    delan3d.pasx=0;
    delan3d.pasy=0;
    delan3d.up=0;
    delan3d.left=0;
    delan3d.RyaE=0;
    delan3d.RyaV=0;
    delan3d.RyaH=-2;
    delan3d.Theta=0;
    delan3d.Rmult=0;
    return TRUE;
}

////////////////////////////////////////
// FONCTION     delan_3d_destroy
// DESCRIPTION  Destructeur de la fenetre 3D
// PARAMETRES   hWnd: handler  de la fenetre 3D
// RETOUR       Aucun
////////////////////////////////////////

void delan_3d_destroy (HWND hWnd)
{
    int i;

// Liberation des variables de travail
    for (i=0;i<delan3d.nblist;i++)
    {
        free (delan3d.points[i].x);
        free (delan3d.points[i].y);
        free (delan3d.points[i].z);
    }
    free (delan3d.points);
    delan3d.nblist=0;
// Fermeture de la session OpenGL
    wglDeleteContext(delan3d.ghRC);
    ReleaseDC(hWnd,delan3d.ghDC);
}
```


Mar 25, 01 8:03 d:\ma\tria\ddoctannexes\ddoedit.h Page 1/1
// Fichier moedit.h
LRESULT CALLBACK delan_child_edit(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
void delan_edit_destroy(HWND hWnd);
void delan_edit_keydown(HWND hWnd, int vkey, int tmp, int x, int y);



Mar 25, 01 9:09 d:\maitrise\doc\annexes\ddeedit.c Page 1/2

```
////////////////////////////////////////
// PROGRAMME    Edite d'edition
// BUT          Faire la gestion de la boite d'edition
// FICHIER      deedit.c
// DESCRIPTION  Fait la gestion de la boite d'edition pour
//              les parametres des delta-logarithmes
// REMARQUE     Aucune
// AUTEUR       Nicolas Leduc
// DATE         15 septembre 2000
////////////////////////////////////////

#include <windows.h>
#include <windowsx.h>
#include <math.h>
#include "delan.h"
#include "deedit.h"

// Variable definies dans delan.c
extern HWND edit_win;
extern LPARAM delanparams;

////////////////////////////////////////
// FONCTION     delan_child_edit
// DESCRIPTION  Handler des messages recus par la fenetre d'edition
////////////////////////////////////////

LRESULT CALLBACK delan_child_edit(HWND hWnd, UINT message, WPARAM wParam, LPARAM
 lParam)
{
   switch (message)
   {
      case WM_DESTROY : return HANDLE_WM_DESTROY(hWnd,wParam,lParam,delan_edit
_destroy);
      case WM_KEYDOWN : if (wParam==VK_RETURN||wParam==VK_ESCAPE)
                        return HANDLE_WM_KEYDOWN(hWnd,wParam,lParam,delan_edit_k
eydown);
                        break;
   }
   return CallWindowProc((WNDPROC) GetWindowLong(hWnd,GWL_USERDATA),hWnd,message
,wParam,lParam);
}

////////////////////////////////////////
// FONCTION     delan_edit_keydown
// DESCRIPTION  Fonction qui prends en charge la pression d'une
//              touche dans la fenetre d'edition
// PARAMETRES   hWnd: Handler de fenetre
//              vkey: code de la cle pressee
//              cmp,x,y: inutil.se
// RETOUR       Aucun
////////////////////////////////////////

void delan_edit_keydown(HWND hWnd, int vkey, int cmp, int x, int y)
{
    char buffer[50];
    double temp;

// Si l'usager presse ESC on annule les modification
    if (vkey==VK_ESCAPE)
    {
        DestroyWindow(hWnd);
        return;
    }
// Si l'usager presse ENTER on sauvegarde les modification
    if (vkey==VK_RETURN)
```



Mar 25, 01 9:09 d:\maitrise\doc\annexes\ddeedit.c Page 2/2

```
    {
        Edit_GetText(hWnd,buffer,50);
        temp=atof(buffer);
        delanparams.digna[delanparams.pos*15+delanparams.curseur]=temp;
        DestroyWindow(hWnd);
        return;
    }
}

////////////////////////////////////////
// FONCTION     delan_edit_destroy
// DESCRIPTION  Destructeur de la fenetre d'edition
// PARAMETRES   hWnd: Handler de la fenetre d'edition
// RETOUR       Aucun
////////////////////////////////////////

void delan_edit_destroy(HWND hWnd)
{
    DWORD oldproc;

// On remet l'ancien handler de la fenetre d'edition
    oldproc=GetWindowLong(hWnd,GWL_USERDATA);
    if (oldproc!=0)
        SetWindowLong(hWnd,GWL_WNDPROC,oldproc);
    edit_win=0;
}
```

```
//////////////////////////////////////////
// Fichier delan.h
//////////////////////////////////////////

#include <windows.h>
#include "dduser.h"

#define PI      3.14159265535

// Structure contenant la liste des points
typedef struct
{
    int onflag,nbpts;
    int R,G,B;
    float *X,*Y,*Z;
} LISTEPTS;

// Structure de controle de la fenetre contenant les traces en 3D
typedef struct
{
    HWND hWnd;
        int nblist;
    LISTEPTS *points;
    float cx,cy,zoom;
    int flags,pmax,pmay,up,left;
} WIN3D;

// Structure de controle de la fenetre contenant les traces en 1D
typedef struct
{
    HWND hwnd;
        HDC gADC;
        HGLRC gARC;
    int nblist;
    LISTEPTS *points,
        double Xref,Yref,Zref,
        double Theta,Rouli;
    int flags,pmax,pmay,up,left,
} WIN1D;

// Structure contenant les parametres des courbes delta-logarithmiques
typedef struct
{
    int ndlogn;
    double *dlogna;
    int nseg;
    int *seginfo;
    int editno,pos,curseur,seg,
} WINPARAMS;

// Definition des routines contenues dans delan.c
LRESULT CALLBACK delan_main(HWND hWnd, UINT message, WPARAM wParam, LPARAM lparam);
void AddPoints2d(int,int,int,int,float *,float *,int);
void AddPoints3d(int,int,int,int,float *,float *,float *,int);
void DeletePoints2d(int flag);
void DeletePoints3d(int flag);
void dl_comm(HWND hWnd,int wID,HWND hwndCtl,int wNotifyCode);
int delan_main_create (HWND hWnd,LPCREATESTRUCT lpCs);
void delan_main_paint(HWND hWnd);
void delan_main_destroy (HWND hWnd);
void delan_main_command(HWND hWnd,int wID,HWND hwndCtl,int wNotifyCode);
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst, LPSTR lpCmdLine, in
t nCmdShow);
```

```
////////////////////////////////////////
// PROGRAMME    Visualisation 3D
// BUT          Visualiser les trajectoires et les profils de vitesse
// FICHIER      delan.c
// DESCRIPTION  Programme qui gere la fenetre principale pour
//              l'affichage des trajectoires, des profils et des
//              parametres
// REMARQUE     Utilise Win32 & OpenGL
// AUTEUR       Nicolas Leduc
// DATE         15 aout 2000
////////////////////////////////////////

#include <stdio.h>
#include <window.h>
#include <windowsx.h>
#include <math.h>
#include <malloc.h>
#include "delan.h"
#include "GLmath.h"
#include "mouse.h"
#include "dlan2d.h"
#include "dlan3d.h"
#include "dlan3d.h"
#include "dlan3d.h"

#define itoa _itoa //Pour GCC

// Definition des variables globales
HWND edit_win;
HINSTANCE hInst;
WNDCLASS delan2d;
WNDCLASS delan3d;
WINPARAMS delanparams;

////////////////////////////////////////
// FONCTION     delan_main
// DESCRIPTION  Handler des messages de la fenetre principale
////////////////////////////////////////

LRESULT CALLBACK delan_main(HWND hWnd, UINT message, WPARAM wParam, LPARAM lPara
m)
{
        switch (message)
                {
                        case WM_PAINT   return HANDLE_WM_PAINT(hWnd,wParam,lParam,dela
n_main_paint);
                        case WM_DESTROY  return HANDLE_WM_DESTROY(hWnd,wParam, lPar
am, delan_main_destroy);
                        case WM_CREATE          return HANDLE_WM_CREATE(hWnd,wParam,lP
aram,delan_main_create);
                        case WM_COMMAND  return HANDLE_WM_COMMAND(hWnd,wParam,lPar
am,delan_main_command);
                }
        return DefWindowProc(hWnd,message,wParam,lParam);
}

////////////////////////////////////////
// FONCTION     AddPoints2d
// DESCRIPTION  Ajoute une liste de points a afficher pour la
//              fenetre 2D
// PARAMETRES   flag: defini dans quelle circonstance le trace
//                    doit etre visible
//              R,G,B: Couleur du trace
//              x,y: liste des points
//              nbpts: nombre de points dans les listes
// RETOUR       Aucun
////////////////////////////////////////

void AddPoints2d (int flag,int R,int G,int B,float *x,float *y,int nbpts)
{
        int i;

// Initialise la structure
        delan2d.nblist++;
        delan2d.points=realloc(delan2d.points,delan2d.nblist*sizeof(LISTEPTS));
        delan2d.points[delan2d.nblist-1].nbpts=nbpts;
        delan2d.points[delan2d.nblist-1].R=R;
        delan2d.points[delan2d.nblist-1].G=G;
        delan2d.points[delan2d.nblist-1].B=B;
        delan2d.points[delan2d.nblist-1].onflag=flag;
        delan2d.points[delan2d.nblist-1].x=malloc(nbpts*sizeof(float));
        delan2d.points[delan2d.nblist-1].y=malloc(nbpts*sizeof(float));
// Copie les points
        for (i=0;i<nbpts;i++)
        {
                delan2d.points[delan2d.nblist-1].x[i]=x[i];
                delan2d.points[delan2d.nblist-1].y[i]=y[i];
        }
}

////////////////////////////////////////
// FONCTION     AddPoints3d
// DESCRIPTION  Ajoute une liste de points a afficher dans la
//              fenetre 3D
// PARAMETRES   flag: defini dans quelle circonstance le trace
//                    doit etre visible
//              R,G,B: Couleur du trace
//              x,y,z: liste des points
//              nbpts: nombre de points dans les listes
// RETOUR       Aucun
////////////////////////////////////////

void AddPoints3d (int flag,int R,int G,int B,float *x,float *y,float *z,int nbpt
s)
{
        int i;

// Initialise la structure
        delan3d.nblist++;
        delan3d.points=realloc(delan3d.points,delan3d.nblist*sizeof(LISTEPTS));
        delan3d.points[delan3d.nblist-1].nbpts=nbpts;
        delan3d.points[delan3d.nblist-1].R=R;
        delan3d.points[delan3d.nblist-1].G=G;
        delan3d.points[delan3d.nblist-1].B=B;
        delan3d.points[delan3d.nblist-1].onflag=flag;
        delan3d.points[delan3d.nblist-1].x=malloc(nbpts*sizeof(float));
        delan3d.points[delan3d.nblist-1].y=malloc(nbpts*sizeof(float));
        delan3d.points[delan3d.nblist-1].z=malloc(nbpts*sizeof(float));
// Copie les points
        for (i=0;i<nbpts;i++)
        {
                delan3d.points[delan3d.nblist-1].x[i]=x[i];
                delan3d.points[delan3d.nblist-1].y[i]=y[i];
                delan3d.points[delan3d.nblist-1].z[i]=z[i];
        }
}

////////////////////////////////////////
// FONCTION     DeletePoints2d
// DESCRIPTION  Detruit une liste de points a afficher dans la
//              fenetre 2D
// PARAMETRES   flag: indique la ou les listes a detruire
// RETOUR       Aucun
////////////////////////////////////////

void DeletePoints2d (int flag)
{
    int i,j;

// Pour chacune des listes
    for (i=0;i<delan2d.nblist;i++)
    {
// si le flag en parametre = flag de la liste on la detruit
        if ((delan2d.points[i].onflag&flag)==0)
        {
// Mise a jour de la structure
            delan2d.nblist--;
            delan2d.points[i].x=realloc(delan2d.points[i].x,delan2d.points[delan
2d.nblist].nbpts*sizeof(float));
            delan2d.points[i].y=realloc(delan2d.points[i].y,delan2d.points[delan
2d.nblist].nbpts*sizeof(float));
            delan2d.points[i].nbpts=delan2d.points[delan2d.nblist].nbpts;
            delan2d.points[i].R=delan2d.points[delan2d.nblist].R;
            delan2d.points[i].G=delan2d.points[delan2d.nblist].G;
            delan2d.points[i].B=delan2d.points[delan2d.nblist].B;
            delan2d.points[i].onflag=delan2d.points[delan2d.nblist].onflag;
            for (j=0;j<delan2d.points[i].nbpts;j++)
            {
                delan2d.points[i].x[j]=delan2d.points[delan2d.nblist].x[j];
                delan2d.points[i].y[j]=delan2d.points[delan2d.nblist].y[j];
            }
            free(delan2d.points[delan2d.nblist].x);
            free(delan2d.points[delan2d.nblist].y);
            delan2d.points=realloc(delan2d.points,delan2d.nblist*sizeof(LISTEPTS
));
            i--;
        }
    }
}

////////////////////////////////////////
// FONCTION     DeletePoints3d
// DESCRIPTION  Efface la liste de points des traces a afficher
//              dans la fenetre 3D
// PARAMETRES   flag: indique la ou les listes a detruire
// RETOUR       Aucun
////////////////////////////////////////

void DeletePoints3d(int flag)
{
    int i,j;

// Pour toutes les listes de points
    for (i=0;i<=delan3d.nblist;i++)
    {
// si le flag en parametre = flag de la liste on detruit celle-ci
        if ((delan3d.points[i].onflag&flag)==0)
        {
// Mise a jour de la structure
            delan3d.nblist--;
            delan3d.points[i].x=realloc(delan3d.points[i].x,delan3d.points[delan
3d.nblist].nbpts*sizeof(float));
            delan3d.points[i].y=realloc(delan3d.points[i].y,delan3d.points[delan
3d.nblist].nbpts*sizeof(float));
```



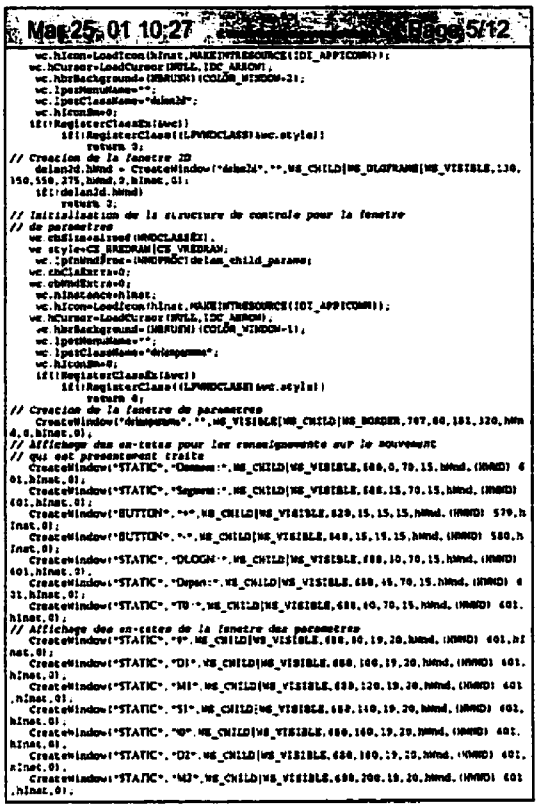
```
//              nbpts: nombre de points dans les listes
// RETOUR       Aucun
////////////////////////////////////////

void AddPoints2d (int flag,int R,int G,int B,float *x,float *y,int nbpts)
{
        int i;

// Initialise la structure
        delan2d.nblist++;
        delan2d.points=realloc(delan2d.points,delan2d.nblist*sizeof(LISTEPTS));
        delan2d.points[delan2d.nblist-1].nbpts=nbpts;
        delan2d.points[delan2d.nblist-1].R=R;
        delan2d.points[delan2d.nblist-1].G=G;
        delan2d.points[delan2d.nblist-1].B=B;
        delan2d.points[delan2d.nblist-1].onflag=flag;
        delan2d.points[delan2d.nblist-1].x=malloc(nbpts*sizeof(float));
        delan2d.points[delan2d.nblist-1].y=malloc(nbpts*sizeof(float));
// Copie les points
        for (i=0;i<nbpts;i++)
        {
                delan2d.points[delan2d.nblist-1].x[i]=x[i];
                delan2d.points[delan2d.nblist-1].y[i]=y[i];
        }
}

////////////////////////////////////////
// FONCTION     AddPoints3d
// DESCRIPTION  Ajoute une liste de points a afficher dans la
//              fenetre 3D
// PARAMETRES   flag: defini dans quelle circonstance le trace
//                    doit etre visible
//              R,G,B: Couleur du trace
//              x,y,z: liste des points
//              nbpts: nombre de points dans les listes
// RETOUR       Aucun
////////////////////////////////////////

void AddPoints3d (int flag,int R,int G,int B,float *x,float *y,float *z,int nbpt
s)
{
        int i;

// Initialise la structure
        delan3d.nblist++;
        delan3d.points=realloc(delan3d.points,delan3d.nblist*sizeof(LISTEPTS));
        delan3d.points[delan3d.nblist-1].nbpts=nbpts;
        delan3d.points[delan3d.nblist-1].R=R;
        delan3d.points[delan3d.nblist-1].G=G;
        delan3d.points[delan3d.nblist-1].B=B;
        delan3d.points[delan3d.nblist-1].onflag=flag;
        delan3d.points[delan3d.nblist-1].x=malloc(nbpts*sizeof(float));
        delan3d.points[delan3d.nblist-1].y=malloc(nbpts*sizeof(float));
        delan3d.points[delan3d.nblist-1].z=malloc(nbpts*sizeof(float));
// Copie les points
        for (i=0;i<nbpts;i++)
        {
                delan3d.points[delan3d.nblist-1].x[i]=x[i];
                delan3d.points[delan3d.nblist-1].y[i]=y[i];
                delan3d.points[delan3d.nblist-1].z[i]=z[i];
        }
}

////////////////////////////////////////
// FONCTION     DeletePoints2d
```



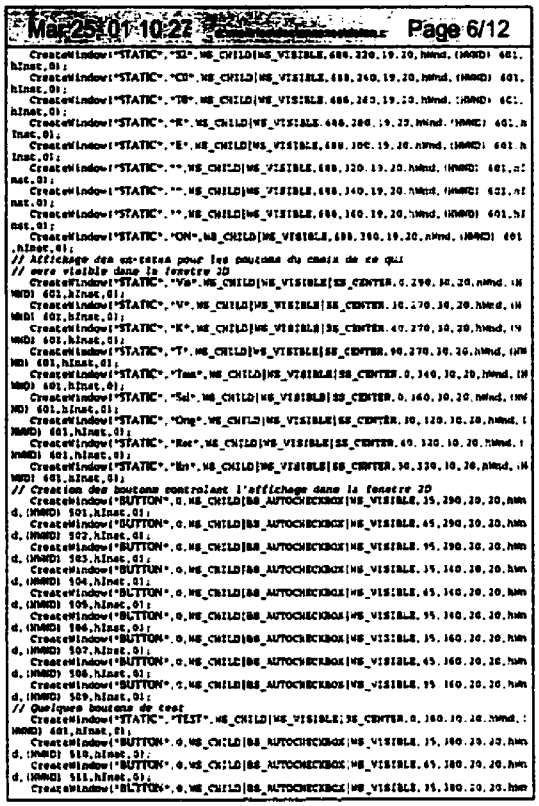
```
// DESCRIPTION  Detruit une liste de points a afficher dans la
//              fenetre 2D
// PARAMETRES   flag: indique la ou les listes a detruire
// RETOUR       Aucun
////////////////////////////////////////

void DeletePoints2d (int flag)
{
    int i,j;

// Pour chacune des listes
    for (i=0;i<delan2d.nblist;i++)
    {
// si le flag en parametre = flag de la liste on la detruit
        if ((delan2d.points[i].onflag&flag)==0)
        {
// Mise a jour de la structure
            delan2d.nblist--;
            delan2d.points[i].x=realloc(delan2d.points[i].x,delan2d.points[delan
2d.nblist].nbpts*sizeof(float));
            delan2d.points[i].y=realloc(delan2d.points[i].y,delan2d.points[delan
2d.nblist].nbpts*sizeof(float));
            delan2d.points[i].nbpts=delan2d.points[delan2d.nblist].nbpts;
            delan2d.points[i].R=delan2d.points[delan2d.nblist].R;
            delan2d.points[i].G=delan2d.points[delan2d.nblist].G;
            delan2d.points[i].B=delan2d.points[delan2d.nblist].B;
            delan2d.points[i].onflag=delan2d.points[delan2d.nblist].onflag;
            for (j=0;j<delan2d.points[i].nbpts;j++)
            {
                delan2d.points[i].x[j]=delan2d.points[delan2d.nblist].x[j];
                delan2d.points[i].y[j]=delan2d.points[delan2d.nblist].y[j];
            }
            free(delan2d.points[delan2d.nblist].x);
            free(delan2d.points[delan2d.nblist].y);
            delan2d.points=realloc(delan2d.points,delan2d.nblist*sizeof(LISTEPTS
));
            i--;
        }
    }
}

////////////////////////////////////////
// FONCTION     DeletePoints3d
// DESCRIPTION  Efface la liste de points des traces a afficher
//              dans la fenetre 3D
// PARAMETRES   flag: indique la ou les listes a detruire
// RETOUR       Aucun
////////////////////////////////////////

void DeletePoints3d(int flag)
{
    int i,j;

// Pour toutes les listes de points
    for (i=0;i<=delan3d.nblist;i++)
    {
// si le flag en parametre = flag de la liste on detruit celle-ci
        if ((delan3d.points[i].onflag&flag)==0)
        {
// Mise a jour de la structure
            delan3d.nblist--;
            delan3d.points[i].x=realloc(delan3d.points[i].x,delan3d.points[delan
3d.nblist].nbpts*sizeof(float));
            delan3d.points[i].y=realloc(delan3d.points[i].y,delan3d.points[delan
3d.nblist].nbpts*sizeof(float));
```



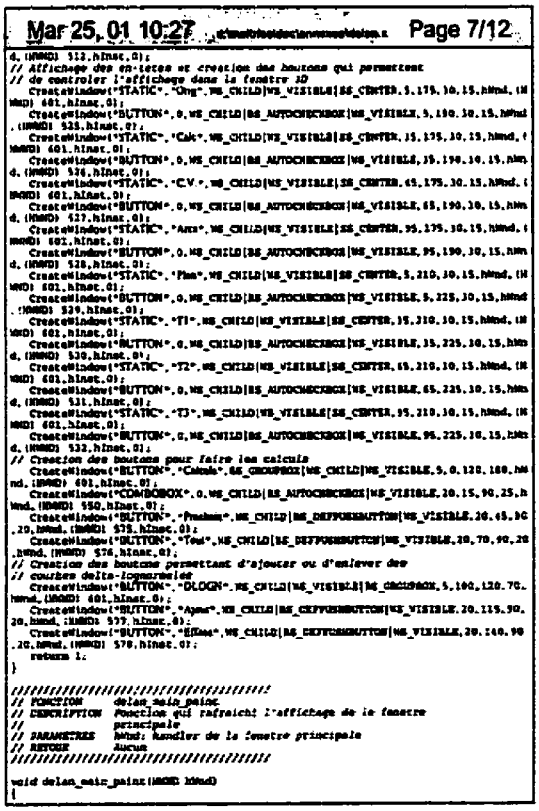
```
            delan3d.points[i].z=realloc(delan3d.points[i].z,delan3d.points[delan
3d.nblist].nbpts*sizeof(float));
            delan3d.points[i].nbpts=delan3d.points[delan3d.nblist].nbpts;
            delan3d.points[i].R=delan3d.points[delan3d.nblist].R;
            delan3d.points[i].G=delan3d.points[delan3d.nblist].G;
            delan3d.points[i].B=delan3d.points[delan3d.nblist].B;
            delan3d.points[i].onflag=delan3d.points[delan3d.nblist].onflag;
            for (j=0;j<delan3d.points[i].nbpts;j++)
            {
                delan3d.points[i].x[j]=delan3d.points[delan3d.nblist].x[j];
                delan3d.points[i].y[j]=delan3d.points[delan3d.nblist].y[j];
                delan3d.points[i].z[j]=delan3d.points[delan3d.nblist].z[j];
            }
            free(delan3d.points[delan3d.nblist].x);
            free(delan3d.points[delan3d.nblist].y);
            free(delan3d.points[delan3d.nblist].z);
            delan3d.points=realloc(delan3d.points,delan3d.nblist*sizeof(LISTEPTS
));
            i--;
        }
    }
}

////////////////////////////////////////
// FONCTION     delan_main_create
// DESCRIPTION  Handler de la creation de la fenetre
// PARAMETRES   hWnd: handler de la fenetre principale
//              lpCs: structure contenant des renseignements
//                    pour la creation de la fenetre
// RETOUR       int. 1 si tout a fonctionne, 0 sinon
////////////////////////////////////////

int delan_main_create (HWND hWnd,LPCREATESTRUCT lpCs)
{
    WNDCLASSEX wc;

// Initialisation de la structure de controle pour la fenetre 3D
    wc.cbSize=sizeof(WNDCLASSEX);
    wc.style=CS_HREDRAW|CS_VREDRAW;
    wc.lpfnWndProc=(WNDPROC)delan_child_3d;
    wc.cbClsExtra=0;
    wc.cbWndExtra=0;
    wc.hInstance=hInst;
    wc.hIcon=LoadIcon(hInst,MAKEINTRESOURCE(IDI_APPICON));
    wc.hCursor=LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+2);
    wc.lpszMenuName="";
    wc.lpszClassName="delan3d";
    wc.hIconSm=0;
    if(!RegisterClassEx(&wc))
        if(!RegisterClass((LPWNDCLASS)&wc.style))
            return 0;
// Creation de la fenetre 3D
    delan3d.hWnd = CreateWindow("delan3d","",WS_OLDFRAME|WS_CHILD|WS_VISIBLE,130,
0,550,350,
    hWnd,(HMENU) 2,hInst,0);
    if(!delan3d.hWnd)
        return 0;
// Initialisation de la structure de controle pour la fenetre 2D
    wc.cbSize=sizeof(WNDCLASSEX);
    wc.style=CS_HREDRAW|CS_VREDRAW;
    wc.lpfnWndProc=(WNDPROC)delan_child_2d;
    wc.cbClsExtra=0;
    wc.cbWndExtra=0;
    wc.hInstance=hInst;
```



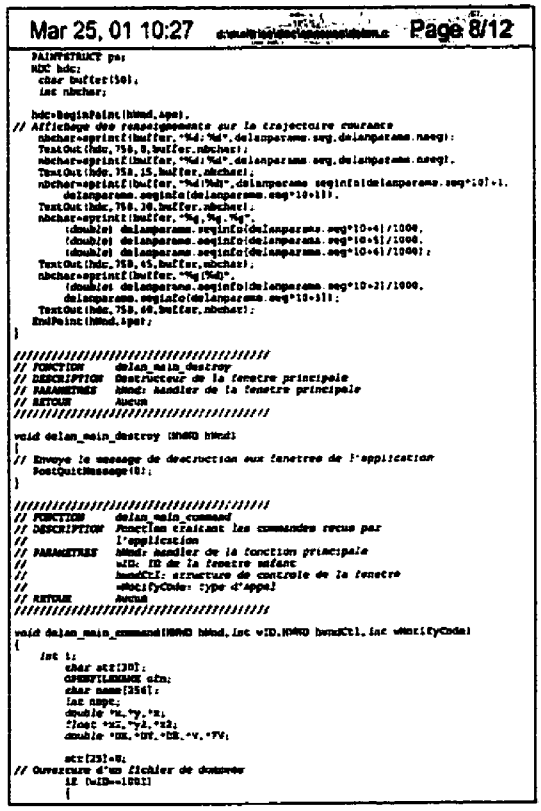
```
Mar 25, 01 10:27                                   Page 5/12

    wc.hIcon=LoadIcon(hInst,MAKEINTRESOURCE(IDI_APPICONN));
    wc.hCursor=LoadCursor(NULL,IDC_ARROW);
    wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+2);
    wc.lpszMenuName="";
    wc.lpszClassName="delan3d";
    wc.hIconSm=0;
    if(!RegisterClassEx(&wc))
        if(!RegisterClass((LPWNDCLASS)&wc.style))
            return 3;
// Creation de la fenetre 3D
    delan3d.hWnd = CreateWindow("delan3d","",WS_CHILD|WS_OLDFRAME|WS_VISIBLE,130,
150,550,375,hWnd,0,hInst,0);
    if(!delan3d.hWnd)
        return 3;
// Initialisation de la structure de controle pour la fenetre
// de parametres
    wc.cbClsExtra=(WNDCLASSEX);
    wc.style=CS_HREDRAW|CS_VREDRAW;
    wc.lpfnWndProc=(WNDPROC)delan_child_params;
    wc.cbClsExtra=0;
    wc.cbWndExtra=0;
    wc.hInstance=hInst;
    wc.hIcon=LoadIcon(hInst,MAKEINTRESOURCE(IDI_APPICONN));
    wc.hCursor=LoadCursor(NULL,IDC_ARROW);
    wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName="";
    wc.lpszClassName="drapeparms";
    wc.hIconSm=0;
    if(!RegisterClassEx(&wc))
        if(!RegisterClass((LPWNDCLASS)&wc.style))
            return 4;
// Creation de la fenetre de parametres
    CreateWindow("drapeparms","",WS_VISIBLE|WS_CHILD|WS_BORDER,707,60,181,320,hWn
d,0,hInst,0);
// Affichage des en-tetes pour les renseignements sur le nouveau
// qui est presentement traite
    CreateWindow("STATIC","Gamma:",WS_CHILD|WS_VISIBLE,588,0,70,15,hWnd,(HMENU) 6
01,hInst,0);
    CreateWindow("STATIC","Sigma:",WS_CHILD|WS_VISIBLE,588,15,70,15,hWnd,(HMENU)
601,hInst,0);
    CreateWindow("BUTTON","+",WS_CHILD|WS_VISIBLE,659,15,15,15,hWnd,(HMENU) 579,h
Inst,0);
    CreateWindow("BUTTON","-",WS_CHILD|WS_VISIBLE,648,15,15,15,hWnd,(HMENU) 580,h
Inst,0);
    CreateWindow("STATIC","DLOGN:",WS_CHILD|WS_VISIBLE,588,30,70,15,hWnd,(HMENU)
601,hInst,0),
    CreateWindow("STATIC","Depn:",WS_CHILD|WS_VISIBLE,588,45,70,15,hWnd,(HMENU) 6
01,hInst,0);
    CreateWindow("STATIC","T0:",WS_CHILD|WS_VISIBLE,588,60,70,15,hWnd,(HMENU) 601.
hInst,0);
// Affichage des en-tetes de la fenetre des parametres
    CreateWindow("STATIC","P",WS_CHILD|WS_VISIBLE,688,80,19,20,hWnd,(HMENU) 601,hI
nst,0);
    CreateWindow("STATIC","D1",WS_CHILD|WS_VISIBLE,688,100,19,20,hWnd,(HMENU) 601.
hInst,0),
    CreateWindow("STATIC","M1",WS_CHILD|WS_VISIBLE,688,120,19,20,hWnd,(HMENU) 601
,hInst,0),
    CreateWindow("STATIC","S1",WS_CHILD|WS_VISIBLE,688,140,19,20,hWnd,(HMENU) 601,
hInst,0),
    CreateWindow("STATIC","D",WS_CHILD|WS_VISIBLE,688,160,19,20,hWnd,(HMENU) 601.
hInst,0),
    CreateWindow("STATIC","D2",WS_CHILD|WS_VISIBLE,688,180,19,20,hWnd,(HMENU) 601.
hInst,0),
    CreateWindow("STATIC","M2",WS_CHILD|WS_VISIBLE,688,200,19,20,hWnd,(HMENU) 601
,hInst,0),
```



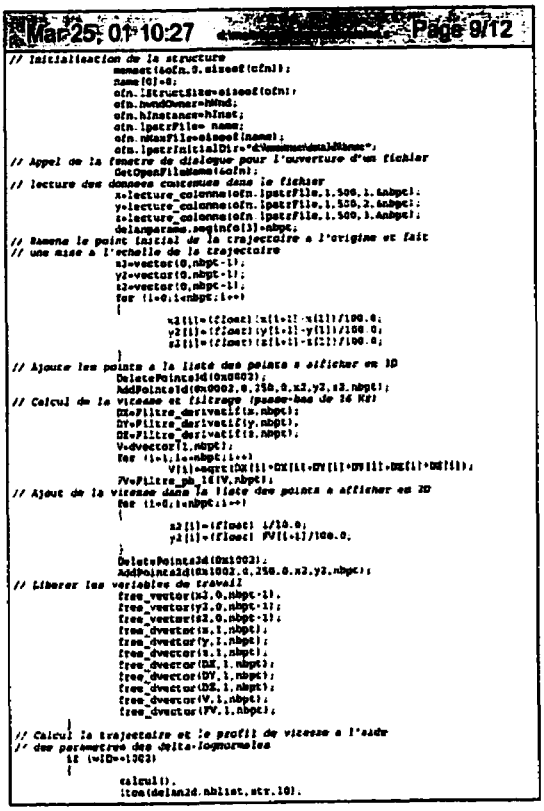
```
Mar 25, 01 10:27                                   Page 6/12

    CreateWindow("STATIC","S2",WS_CHILD|WS_VISIBLE,688,220,19,20,hWnd,(HMENU) 601,
hInst,0);
    CreateWindow("STATIC","D3",WS_CHILD|WS_VISIBLE,688,240,19,20,hWnd,(HMENU) 601,
hInst,0);
    CreateWindow("STATIC","T0",WS_CHILD|WS_VISIBLE,688,260,19,20,hWnd,(HMENU) 601.
hInst,0);
    CreateWindow("STATIC","R",WS_CHILD|WS_VISIBLE,688,280,19,20,hWnd,(HMENU) 601.h
Inst,0);
    CreateWindow("STATIC","E",WS_CHILD|WS_VISIBLE,688,300,19,20,hWnd,(HMENU) 601.h
Inst,0);
    CreateWindow("STATIC","",WS_CHILD|WS_VISIBLE,688,320,19,20,hWnd,(HMENU) 601,hI
nst,0);
    CreateWindow("STATIC","",WS_CHILD|WS_VISIBLE,688,340,19,20,hWnd,(HMENU) 601,hI
nst,0);
    CreateWindow("STATIC","",WS_CHILD|WS_VISIBLE,688,360,19,20,hWnd,(HMENU) 601,hI
nst,0);
    CreateWindow("STATIC","ON",WS_CHILD|WS_VISIBLE,688,380,19,20,hWnd,(HMENU) 601
,hInst,0);
// Affichage des en-tetes pour les boutons du choix de ce qui
// sera visible dans la fenetre 3D
    CreateWindow("STATIC","Va",WS_CHILD|WS_VISIBLE|SS_CENTER,0,290,30,20,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("STATIC","V",WS_CHILD|WS_VISIBLE|SS_CENTER,30,270,30,20,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("STATIC","K",WS_CHILD|WS_VISIBLE|SS_CENTER,60,270,30,20,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("STATIC","T",WS_CHILD|WS_VISIBLE|SS_CENTER,90,270,30,20,hWnd,(HM
MENU) 601,hInst,0);
    CreateWindow("STATIC","Tau",WS_CHILD|WS_VISIBLE|SS_CENTER,0,340,30,20,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("STATIC","Sal",WS_CHILD|WS_VISIBLE|SS_CENTER,0,360,30,20,hWnd,(HM
MENU) 601,hInst,0);
    CreateWindow("STATIC","Ong",WS_CHILD|WS_VISIBLE|SS_CENTER,30,320,30,20,hWnd,(
HMENU) 601,hInst,0);
    CreateWindow("STATIC","Rac",WS_CHILD|WS_VISIBLE|SS_CENTER,60,320,30,20,hWnd,(
HMENU) 601,hInst,0);
    CreateWindow("STATIC","Hn",WS_CHILD|WS_VISIBLE|SS_CENTER,90,320,30,20,hWnd,(H
MENU) 601,hInst,0);
// Creation des boutons controlant l'affichage dans la fenetre 3D
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,35,290,20,20,hWn
d,(HMENU) 501,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,65,290,20,20,hWn
d,(HMENU) 502,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,95,290,20,20,hWn
d,(HMENU) 503,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,35,340,20,20,hWn
d,(HMENU) 504,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,65,340,20,20,hWn
d,(HMENU) 505,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,95,340,20,20,hWn
d,(HMENU) 506,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,35,360,20,20,hWn
d,(HMENU) 507,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILE|BS_AUTOCHECKBOX|WS_VISIBLE,65,360,20,20,hWn
d,(HMENU) 508,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,95,360,20,20,hWn
d,(HMENU) 509,hInst,0);
// Quelques boutons de test
    CreateWindow("STATIC","TEST",WS_CHILD|WS_VISIBLE|SS_CENTER,0,380,30,20,hWnd,(
HMENU) 601,hInst,0),
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,35,380,20,20,hWn
d,(HMENU) 510,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,65,380,20,20,hWn
d,(HMENU) 511,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,95,380,20,20,hWn
```



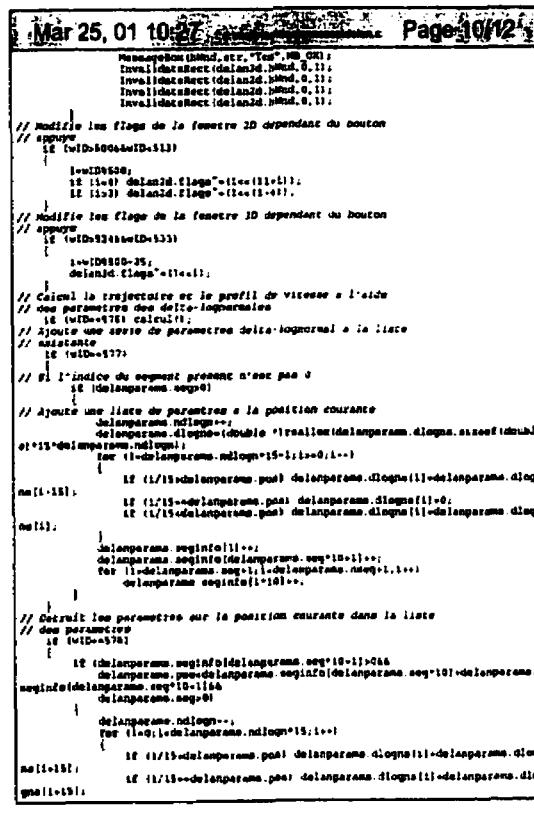
```
Mar 25, 01 10:27                                   Page 7/12

d,(HMENU) 512,hInst,0);
// Affichage des en-tetes et creation des boutons qui permettront
// de controler l'affichage dans la fenetre 3D
    CreateWindow("STATIC","Ong",WS_CHILD|WS_VISIBLE|SS_CENTER,5,175,30,15,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,5,190,30,15,hWnd
,(HMENU) 522,hInst,0),
    CreateWindow("STATIC","Cak",WS_CHILD|WS_VISIBLE|SS_CENTER,35,175,30,15,hWnd,(
HMENU) 601,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,35,190,30,15,hWn
d,(HMENU) 524,hInst,0);
    CreateWindow("STATIC","C.V.",WS_CHILD|WS_VISIBLE|SS_CENTER,65,175,30,15,hWnd,(
HMENU) 601,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,65,190,30,15,hWn
d,(HMENU) 527,hInst,0);
    CreateWindow("STATIC","Arc",WS_CHILD|WS_VISIBLE|SS_CENTER,95,175,30,15,hWnd,(
HMENU) 602,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,95,190,30,15,hWn
d,(HMENU) 528,hInst,0);
    CreateWindow("STATIC","Pas",WS_CHILD|WS_VISIBLE|SS_CENTER,5,210,30,15,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,5,225,30,15,hWnd
,(HMENU) 529,hInst,0);
    CreateWindow("STATIC","T1",WS_CHILD|WS_VISIBLE|SS_CENTER,35,210,30,15,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,35,225,30,15,hWn
d,(HMENU) 530,hInst,0);
    CreateWindow("STATIC","T2",WS_CHILD|WS_VISIBLE|SS_CENTER,65,210,30,15,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,65,225,30,15,hWn
d,(HMENU) 531,hInst,0);
    CreateWindow("STATIC","T3",WS_CHILD|WS_VISIBLE|SS_CENTER,95,210,30,15,hWnd,(H
MENU) 601,hInst,0);
    CreateWindow("BUTTON",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,95,225,30,15,hWn
d,(HMENU) 532,hInst,0);
// Creation des boutons pour faire les calculs
    CreateWindow("BUTTON","Calculs",BS_GROUPBOX|WS_CHILD|WS_VISIBLE,5,0,120,180,hWn
nd,(HMENU) 601,hInst,0);
    CreateWindow("COMBOBOX",0,WS_CHILD|BS_AUTOCHECKBOX|WS_VISIBLE,20,15,90,25,h
Wnd,(HMENU) 550,hInst,0);
    CreateWindow("BUTTON","Problem",WS_CHILD|BS_DEFPUSHBUTTON|WS_VISIBLE,20,45,90
,20,hWnd,(HMENU) 575,hInst,0);
    CreateWindow("BUTTON","Tau",WS_CHILD|BS_DEFPUSHBUTTON|WS_VISIBLE,20,70,90,20
,hWnd,(HMENU) 576,hInst,0);
// Creation des boutons permettant d'ajouter ou d'enlever des
// courbes delta-isopycnales
    CreateWindow("BUTTON","DLOGN",WS_CHILD|WS_VISIBLE|BS_CHECKBOX,5,100,120,70,
hWnd,(HMENU) 601,hInst,0);
    CreateWindow("BUTTON","Agmn",WS_CHILD|BS_DEFPUSHBUTTON|WS_VISIBLE,20,115,90,
20,hWnd,(HMENU) 577,hInst,0);
    CreateWindow("BUTTON","Effac",WS_CHILD|BS_DEFPUSHBUTTON|WS_VISIBLE,20,140,90
,20,hWnd,(HMENU) 578,hInst,0);
    return 1;
}

////////////////////////////////////////
// FONCTION     delan_main_paint
// DESCRIPTION  Fonction qui rafraichi l'affichage de la fenetre
//              principale
// PARAMETRES   hWnd: handler de la fenetre principale
// RETOUR       Aucun
////////////////////////////////////////

void delan_main_paint(HWND hWnd)
{
```



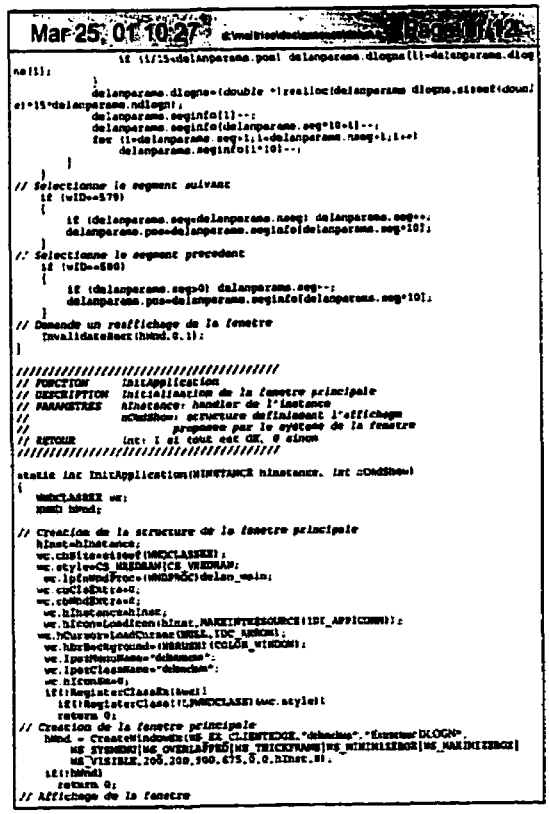
```
Mar 25, 01 10:27                                   Page 8/12

    PAINTSTRUCT ps;
    HDC hdc;
    char buffer[50];
    int nbchar;

    hdc=BeginPaint(hWnd,&ps);
// Affichage des renseignements sur la trajectoire courante
    nbchar=sprintf(buffer,"%d;%d",delanparams.sxg,delanparams.nseg);
    TextOut(hdc,750,0,buffer,nbchar);
    nbchar=sprintf(buffer,"%d;%d",delanparams.sxg,delanparams.nseg);
    TextOut(hdc,750,15,buffer,nbchar);
    nbchar=sprintf(buffer,"%d|%d",delanparams.seqinfo[delanparams.seg*10+0]+1,
           delanparams.seqinfo[delanparams.seg*10+1]);
    TextOut(hdc,750,30,buffer,nbchar);
    nbchar=sprintf(buffer,"%g,%g,%g",
           (double)delanparams.seqinfo[delanparams.seg*10+4]/1000,
           (double)delanparams.seqinfo[delanparams.seg*10+5]/1000,
           (double)delanparams.seqinfo[delanparams.seg*10+6]/1000);
    TextOut(hdc,750,45,buffer,nbchar);
    nbchar=sprintf(buffer,"%g(%d)",
           (double)delanparams.seqinfo[delanparams.seg*10+2]/1000,
           delanparams.seqinfo[delanparams.seg*10+3]);
    TextOut(hdc,750,60,buffer,nbchar);
    EndPaint(hWnd,&ps);
}

////////////////////////////////////////
// FONCTION     delan_main_destroy
// DESCRIPTION  Destructeur de la fenetre principale
// PARAMETRES   hWnd: handler de la fenetre principale
// RETOUR       Aucun
////////////////////////////////////////

void delan_main_destroy (HWND hWnd)
{
// Envoye le message de destruction aux fenetres de l'application
    PostQuitMessage(0);
}

////////////////////////////////////////
// FONCTION     delan_main_command
// DESCRIPTION  Fonction traitant les commandes recus par
//              l'application
// PARAMETRES   hWnd: handler de la fonction principale
//              wID: ID de la fenetre mitant
//              hwndCtl: structure de controle de la fenetre
//              wNotifyCode: type d'appel
// RETOUR       Aucun
////////////////////////////////////////

void delan_main_command(HWND hWnd,int wID,HWND hwndCtl,int wNotifyCode)
{
    int i;
        char str[30];
        OPENFILENAME ofn;
        char name[256];
        int nspc;
        double *X,*Y,*Z;
        float *xz,*yz,*z2;
        double *DX,*DY,*DZ,*V,*TV;

        str[25]=0;
// Ouverture d'un fichier de donnees
        if (wID==1001)
            {
```



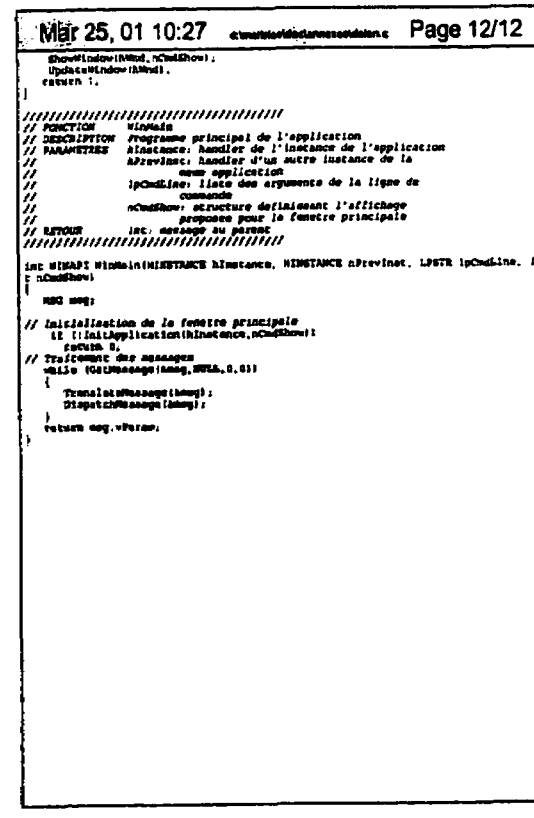
```
Mar 25, 01 10:27                                   Page 9/12
// Initialisation de la structure
                memset(&ofn,0,sizeof(ofn));
                name[0]=0;
                ofn.lStructSize=sizeof(ofn);
                ofn.hwndOwner=hWnd;
                ofn.hInstance=hInst;
                ofn.lpstrFile= name;
                ofn.nMaxFile=sizeof(name);
                ofn.lpstrInitialDir="d:\
// Appel de la fenetre de dialogue pour l'ouverture d'un fichier
                GetOpenFileName(&ofn);
// lecture des donnees contenues dans le fichier
                x=lecture_colonne(ofn.lpstrFile,1,500,1,&nbpt);
                y=lecture_colonne(ofn.lpstrFile,1,500,2,&nbpt);
                z=lecture_colonne(ofn.lpstrFile,1,500,3,&nbpt);
                delanparams.seginfo[3]=nbpt;
// Ramene le point initial de la trajectoire a l'origine et fait
// une mise a l'echelle de la trajectoire
                x2=vector(0,nbpt-1);
                y2=vector(0,nbpt-1);
                z2=vector(0,nbpt-1);
                for (i=0;i<nbpt;i++)
                {
                        x2[i]=(float)(x[i]-x[1])/100.0;
                        y2[i]=(float)(y[i]-y[1])/100.0;
                        z2[i]=(float)(z[i]-z[1])/100.0;
                }
// Ajoute les points a la liste des points a afficher en 3D
                DeletePointsId(0x0803);
                AddPointsId(0x0002,0,256,0,x2,y2,z2,nbpt);
// Calcul de la vitesse et filtrage (passe-bas de 16 Hz)
                DX=Filtre_derivatif(x,nbpt);
                DY=Filtre_derivatif(y,nbpt);
                DZ=Filtre_derivatif(z,nbpt);
                V=dvector(1,nbpt);
                for (i=1;i<=nbpt;i++)
                        V[i]=sqrt(DX[i]*DX[i]+DY[i]*DY[i]+DZ[i]*DZ[i]);
                fV=Filtre_pb_16(V,nbpt);
// Ajout de la vitesse dans la liste des points a afficher en 3D
                for (i=0;i<nbpt;i++)
                {
                        z2[i]=(float) 1/10.0;
                        y2[i]=(float) fV[i+1]/100.0;
                }
                DeletePointsId(0x1003);
                AddPointsId(0x1003,0,256,0,x2,y2,nbpt);
// Liberer les variables de travail
                free_vector(x2,0,nbpt-1);
                free_vector(y2,0,nbpt-1);
                free_vector(z2,0,nbpt-1);
                free_dvector(x,1,nbpt);
                free_dvector(y,1,nbpt);
                free_dvector(z,1,nbpt);
                free_dvector(DX,1,nbpt);
                free_dvector(DY,1,nbpt);
                free_dvector(DZ,1,nbpt);
                free_dvector(V,1,nbpt);
                free_dvector(fV,1,nbpt);
        }
// Calcul la trajectoire et le profil de vitesse a l'aide
// des parametres des delta-lognormales
        if (wID==1303)
        {
                calcul();
                itoa(delan2d.nbliss,str,10);
```



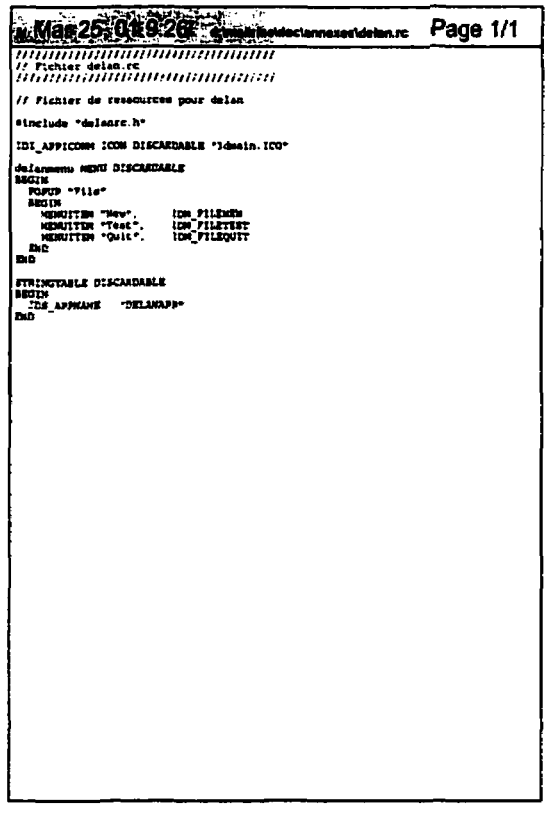
```
Mar 25, 01 10:27                                   Page 10/12
                MessageBox(hWnd,str,"Yes",MB_OK);
                InvalidateRect(delan3d.hWnd,0,1);
                InvalidateRect(delan2d.hWnd,0,1);
                InvalidateRect(delan2d.hWnd,0,1);
                InvalidateRect(delan2d.hWnd,0,1);
        }
// Modifie les flags de la fenetre 2D dependant du bouton
// appuyer
        if (wID>500&&wID<513)
        {
                i=wID%500;
                if (i<9) delan2d.flags^=(1<<(11+i));
                if (i>9) delan2d.flags^=(1<<(i-8));
        }
// Modifie les flags de la fenetre 3D dependant du bouton
// appuyer
        if (wID>524&&wID<533)
        {
                i=(DWORD)wID%25;
                delan3d.flags^=(1<<i);
        }
// Calcul la trajectoire et le profil de vitesse a l'aide
// des parametres des delta-lognormales
        if (wID==578) calcul();
// Ajoute une serie de parametres delta-lognormal a la liste
// existante
        if (wID==577)
        {
// Si l'indice du segment present n'est pas 0
                if (delanparams.seg>0)
                {
// Ajoute une liste de parametres a la position courante
                        delanparams.ndlogn++;
                        delanparams.dlogno=(double *)realloc(delanparams.dlogno,sizeof(doubl
e)*15*delanparams.ndlogn);
                        for (i=delanparams.ndlogn*15-1;i>=0;i--)
                        {
                                if (i/15<delanparams.pos) delanparams.dlogno[i]=delanparams.dlog
no[i-15];
                                if (i/15==delanparams.pos) delanparams.dlogno[i]=0;
                                if (i/15<delanparams.pos) delanparams.dlogno[i]=delanparams.dlog
no[i];
                        }
                        delanparams.seginfo[1]++;
                        delanparams.seginfo[delanparams.seg*10+1]++;
                        for (i=delanparams.seg+1;i<delanparams.nseg+1;i++)
                                delanparams.seginfo[i*10]++;
                }
        }
// Detruit les parametres sur la position courante dans la liste
// des parametres
        if (wID==578)
        {
                if (delanparams.seginfo[delanparams.seg*10+1]>0&&
                        delanparams.pos<delanparams.seginfo[delanparams.seg*10]+delanparams.
seginfo[delanparams.seg*10+1]&&
                        delanparams.seg>0)
                {
                        delanparams.ndlogn--;
                        for (i=0;i<delanparams.ndlogn*15;i++)
                        {
                                if (i/15<delanparams.pos) delanparams.dlogno[i]=delanparams.dlog
no[i+15];
                                if (i/15>=delanparams.pos) delanparams.dlogno[i]=delanparams.dlo
gno[i+15];
```



```
Mar 25, 01 10:27                                   Page 11/12
                if (i/15<delanparams.pos) delanparams.dlogno[i]=delanparams.dlog
no[i];
                }
                delanparams.dlogno=(double *)realloc(delanparams.dlogno,sizeof(doubl
e)*15*delanparams.ndlogn);
                delanparams.seginfo[1]--;
                delanparams.seginfo[delanparams.seg*10+1]--;
                for (i=delanparams.seg+1;i<=delanparams.nseg+1;i++)
                        delanparams.seginfo[i*10]--;
        }
    }
// Selectionne le segment suivant
    if (wID==579)
    {
        if (delanparams.seg<delanparams.nseg) delanparams.seg++;
        delanparams.pos=delanparams.seginfo[delanparams.seg*10];
    }
// Selectionne le segment precedent
    if (wID==580)
    {
        if (delanparams.seg>0) delanparams.seg--;
        delanparams.pos=delanparams.seginfo[delanparams.seg*10];
    }
// Demande un reaffichage de la fenetre
    InvalidateRect(hWnd,0,1);
}

////////////////////////////////////////
// FONCTION     InitApplication
// DESCRIPTION  Initialisation de la fenetre principale
// PARAMETRES   hInstance: handler de l'instance
//              nCmdShow: structure definissant l'affichage
//                        proposee par le systeme de la fenetre
// RETOUR       int: 1 si tout est OK, 0 sinon
////////////////////////////////////////

static int InitApplication(HINSTANCE hInstance, int nCmdShow)
{
   WNDCLASSEX wc;
   HWND hWnd;

// Creation de la structure de la fenetre principale
   hInst=hInstance;
   wc.cbSize=sizeof(WNDCLASSEX);
   wc.style=CS_HREDRAW|CS_VREDRAW;
   wc.lpfnWndProc=(WNDPROC)delan_main;
   wc.cnClsExtra=0;
   wc.cbWndExtra=0;
   wc.hInstance=hInst;
   wc.hIcon=LoadIcon(hInst,MAKEINTRESOURCE(IDI_APPICON));
   wc.hCursor=LoadCursor(NULL,IDC_ARROW);
   wc.hbrBackground=(HBRUSH)(COLOR_WINDOW);
   wc.lpszMenuName="delanmen";
   wc.lpszClassName="delanclass";
   wc.hIconSm=0;
   if(!RegisterClassEx(&wc))
     if(!RegisterClass((LPWNDCLASS)&wc.style))
       return 0;
// Creation de la fenetre principale
   hWnd = CreateWindow(WS_EX_CLIENTEDGE,"delanclass","fenetre DLOGN",
       WS_SYSMENU|WS_OVERLAPPED|WS_THICKFRAME|WS_MINIMIZEBOX|WS_MAXIMIZEBOX|
       WS_VISIBLE,208,208,900,675,0,0,hInst,0);
   if(!hWnd)
       return 0;
// Affichage de la fenetre
```

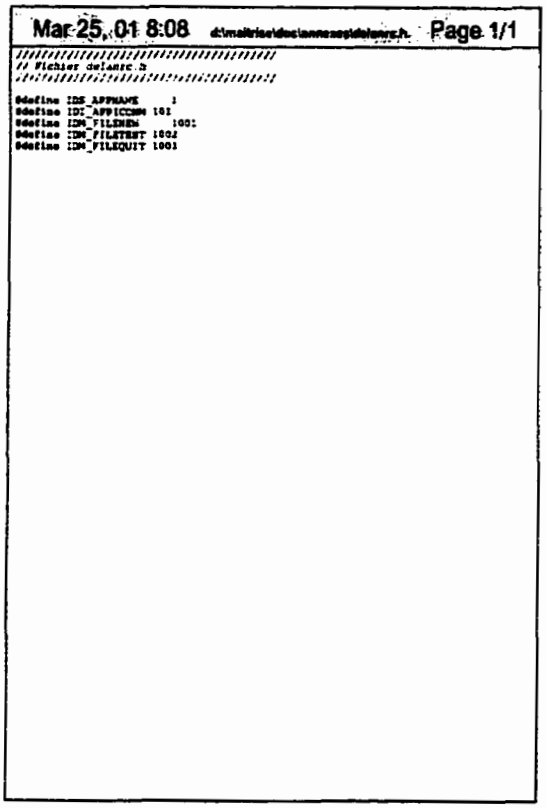


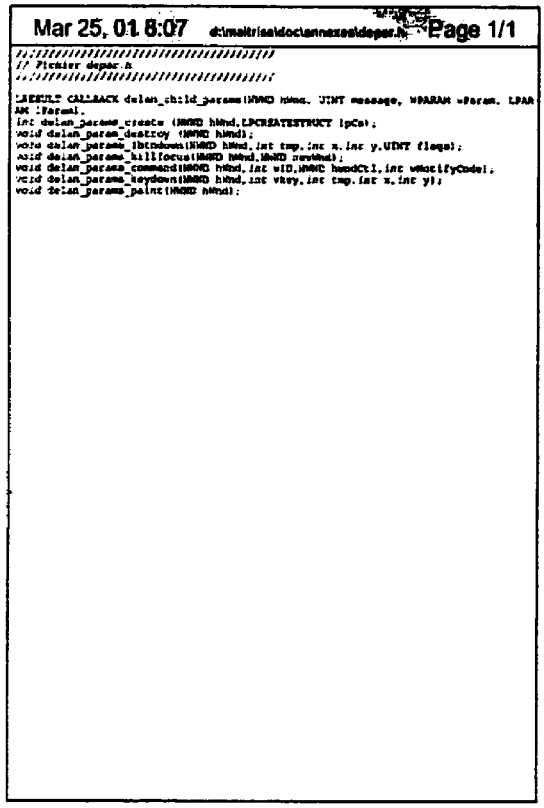
```
Mar 25, 01 10:27                                   Page 12/12
    ShowWindow(hWnd,nCmdShow);
    UpdateWindow(hWnd);
    return 1;
}

////////////////////////////////////////
// FONCTION     WinMain
// DESCRIPTION  Programme principal de l'application
// PARAMETRES   hInstance: handler de l'instance de l'application
//              hPrevInst: handler d'un autre instance de la
//                         meme application
//              lpCmdLine: liste des arguments de la ligne de
//                         commande
//              nCmdShow: structure definissant l'affichage
//                        proposee pour la fenetre principale
// RETOUR       int: message au parent
////////////////////////////////////////

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst, LPSTR lpCmdLine, in
t nCmdShow)
{
   MSG msg;

// Initialisation de la fenetre principale
   if (!InitApplication(hInstance,nCmdShow))
     return 0;
// Traitement des messages
   while (GetMessage(&msg,NULL,0,0))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
   return msg.wParam;
}
```



Mar 25 01:26 C:\Delphi\Exemples\delan\delan.rc Page 1/1

```
//////////////////////////////////////////
// Fichier delan.rc
//////////////////////////////////////////

// Fichier de ressources pour delan

#include "delanrc.h"

IDI_APPICONN ICON DISCARDABLE "idmain.ICO"

delanmenu MENU DISCARDABLE
BEGIN
  POPUP "File"
  BEGIN
    MENUITEM "New",      IDM_FILENEW
    MENUITEM "Test",     IDM_FILETEST
    MENUITEM "Quit",     IDM_FILEQUIT
  END
END

STRINGTABLE DISCARDABLE
BEGIN
  IDS_APPNAME    "DELANAPP"
END
```


Mar 25, 01 8:08 d:\maitrise\doc\annexes\ide\ideinrc.h Page 1/1
// Fichier ideinrc.h
#define IDS_APPNAME 1
#define IDS_APPICCHOM 101
#define IDM_FILENEW 1001
#define IDM_FILETEST 1002
#define IDM_FILEQUIT 1003


Mar 25, 01 8:07 d:\mail\rsa\doc\annexes\depar.h Page 1/1
//////////////////////////////////////
// Fichier depar.h
//////////////////////////////////////
LRESULT CALLBACK delan_child_params(HWND hWnd, UINT message, WPARAM wParam, LPAR
AM lParam);
int delan_params_create (HWND hWnd,LPCREATESTRUCT lpCs);
void delan_param_destroy (HWND hWnd);
void delan_params_lbtndown(HWND hWnd,int tmp,int x,int y,UINT flags);
void delan_params_killfocus(HWND hWnd,HWND newWnd);
void delan_params_command(HWND hWnd,int wID,HWND hwndCtl,int wNotifyCode);
void delan_params_keydown(HWND hWnd,int vkey,int tmp,int x,int y);
void delan_params_paint(HWND hWnd);

```
////////////////////////////////////////
// PROGRAMME    Gestion des parametres
// BUT          Gerer la fenetre d'affichage des parametres
// FICHIER      depar.c
// DESCRIPTION  Gere la fenetre des parametres delta-logarithmique et
//              permet de les modifier
// REMARQUE     Aucune
// AUTEUR       Nicolas Leduc
// DATE         18 aout 2000
////////////////////////////////////////

#include <windows.h>
#include <windowsx.h>
#include <stdio.h>
#include <malloc.h>
#include "delan.h"
#include "depar.h"
#include "draw.h"

// Variable globale de delan.c
extern HWND edit_win;
extern HINSTANCE hInst;
extern WINPARAMS delanparams;

//////////////////////////////////////////
// FONCTION     delan_child_params
// DESCRIPTION  Handler de messages pour la fenetre des parametres
//////////////////////////////////////////

LRESULT CALLBACK delan_child_params(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_PAINT   : return HANDLE_WM_PAINT(hWnd,wParam,lParam,delan_params_paint);
        case WM_DESTROY : return HANDLE_WM_DESTROY(hWnd,wParam,lParam,delan_params_destroy);
        case WM_LBUTTONDOWN : return HANDLE_WM_LBUTTONDOWN(hWnd,wParam,lParam,delan_params_lbtndown);
        case WM_CREATE  : return HANDLE_WM_CREATE(hWnd,wParam,lParam,delan_params_create);
        case WM_KEYDOWN : return HANDLE_WM_KEYDOWN(hWnd,wParam,lParam,delan_params_keydown);
        case WM_COMMAND : return HANDLE_WM_COMMAND(hWnd,wParam,lParam,delan_params_command);
        case WM_KILLFOCUS : return HANDLE_WM_KILLFOCUS(hWnd,wParam,lParam,delan_params_killfocus);
    }
    return DefWindowProc(hWnd,message,wParam,lParam);
}

//////////////////////////////////////////
// FONCTION     delan_params_create
// DESCRIPTION  Creation de la fenetre des parametres
// PARAMETRES   hWnd: handler de la fenetre parametres
//              lpCs: structure de creation de la fenetre
// RETOUR       int: 1 si OK, 0 sinon
//////////////////////////////////////////

int delan_params_create (HWND hWnd,LPCREATESTRUCT lpCs)
{
    int i;

// Initialisation des structures et variables
```



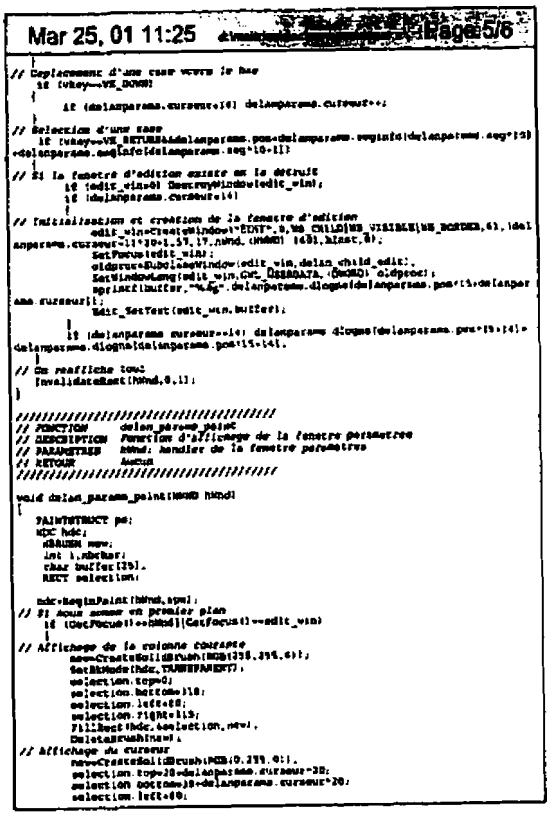
```
    delanparams.pos=0;
    delanparams.curseur=0;
    delanparams.seg=0;
    delanparams.dlogns=(double *) malloc(25*15*sizeof(double));
    for (i=0;i<25*15;i++) delanparams.dlogns[i]=i*1.000012;
    delanparams.ndlogn=25;
    delanparams.seginfo=(int *) malloc(6*10*sizeof(int));
    delanparams.nseg=6;
    delanparams.dlogns[0]=15.5;
    delanparams.dlogns[1]=-0.61;
    delanparams.dlogns[2]=0.10;
    delanparams.dlogns[3]=-0.13;
    delanparams.dlogns[4]=-6.145;
    delanparams.dlogns[5]=-0.44;
    delanparams.dlogns[6]=0.13;
    delanparams.dlogns[7]=0;
    delanparams.dlogns[8]=0;
    delanparams.dlogns[9]=0;
    delanparams.dlogns[10]=0;
    delanparams.seginfo[0]=0;
    delanparams.seginfo[1]=25;
    delanparams.seginfo[2]=0;
    delanparams.seginfo[3]=1000;
    delanparams.seginfo[4]=delanparams.seginfo[5]=delanparams.seginfo[6]=0;
    delanparams.seginfo[7]=delanparams.seginfo[8]=delanparams.seginfo[9]=0;
    delanparams.seginfo[10]=0;
    delanparams.seginfo[11]=0;
    delanparams.seginfo[12]=250;
    delanparams.seginfo[13]=delanparams.seginfo[14]=delanparams.seginfo[15]=delanparams.seginfo[16]=delanparams.seginfo[17]=delanparams.seginfo[18]=delanparams.seginfo[19]=0;
    delanparams.seginfo[20]=5;
    delanparams.seginfo[21]=0;
    delanparams.seginfo[22]=250;
    delanparams.seginfo[23]=250;
    delanparams.seginfo[24]=-1;
    delanparams.seginfo[25]=0;
    delanparams.seginfo[26]=1;
    delanparams.seginfo[27]=delanparams.seginfo[28]=delanparams.seginfo[29]=0;
    delanparams.seginfo[30]=10;
    delanparams.seginfo[31]=5;
    delanparams.seginfo[32]=100;
    delanparams.seginfo[33]=250;
    delanparams.seginfo[34]=1;
    delanparams.seginfo[35]=0;
    delanparams.seginfo[36]=-1;
    delanparams.seginfo[37]=delanparams.seginfo[38]=delanparams.seginfo[39]=0;
    delanparams.seginfo[40]=15;
    delanparams.seginfo[41]=5;
    delanparams.seginfo[42]=250;
    delanparams.seginfo[43]=250;
    delanparams.seginfo[44]=delanparams.seginfo[45]=delanparams.seginfo[46]=1;
    delanparams.seginfo[47]=delanparams.seginfo[48]=delanparams.seginfo[49]=0;
    delanparams.seginfo[50]=20;
    delanparams.seginfo[51]=1;
    delanparams.seginfo[52]=1000;
    delanparams.seginfo[53]=250;
    delanparams.seginfo[54]=delanparams.seginfo[55]=delanparams.seginfo[56]=-1;
    delanparams.seginfo[57]=delanparams.seginfo[58]=delanparams.seginfo[59]=2;
    return 1;
}

//////////////////////////////////////////
// FONCTION     delan_params_destroy
// DESCRIPTION  Destructeur de la fenetre parametres
```



```
// PARAMETRES   hWnd: handler de la fenetre parametres
// RETOUR       Aucun
//////////////////////////////////////////

void delan_params_destroy (HWND hWnd)
{
// Libere les variables de travail
    free (delanparams.dlogns);
    free (delanparams.seginfo);
    delanparams.nseg=0;
    delanparams.ndlogn=0;
}

//////////////////////////////////////////
// FONCTION     delan_params_lbtndown
// DESCRIPTION  Gestion de la pression du bouton gauche de la souris
// PARAMETRES   hWnd: handler de la fenetre parametres
//              tmp: inutilise
//              x,y: position de la souris
//              flags: statut de la souris
// RETOUR       Aucun
//////////////////////////////////////////

void delan_params_lbtndown(HWND hWnd,int tmp,int x,int y,UINT flags)
{
    int posx, posy;
    WNDPROC oldproc;
    char buffer[50];

// Initialisation des variables
    posx=x/60;
    posy=y/20;
// Si on etait en mode edition, on detruit la fenetre
    if (edit_win!=0) DestroyWindow(edit_win);
    SetFocus(hWnd);
// Si on a pas selectionne une case vide
    if (posy>delanparams.nseg-1||posx>delanparams.seginfo[delanparams.seg*10]||posx<delanparams.pos+delanparams.seginfo[delanparams.seg*10+delanparams.seginfo[delanparams.seg*10+1]+1])
    {
// On positionne la colonne choisie au centre de la fenetre
        delanparams.pos=posx-1+delanparams.pos;
        delanparams.curseur=posy-1;
        posx=1;
// Si on a choisi une case editable
        if (posy<15)
        {
// On active la fenetre d'edition
            edit_win=CreateWindow("EDIT","0",WS_CHILD|WS_VISIBLE|WS_BORDER,posx*60+1,posy*20+1,57,17,hWnd,(HMENU) 1601,hInst,0);
            SetFocus(edit_win);
            oldproc=SubclassWindow(edit_win,delan_child_edit);
            SetWindowLong(edit_win,GWL_USERDATA,(DWORD) oldproc);
            sprintf(buffer,"%.6g",delanparams.dlogns[(delanparams.pos+posx-1)*15+posy-1]);
            Edit_SetText(edit_win,buffer);
        }
        if (posy==15) delanparams.dlogns[delanparams.pos*15+14]=delanparams.dlogns[delanparams.pos*15+14];
    }
    if (posy==delanparams.nseg||posx>delanparams.seginfo[delanparams.seg*10]&&posx<delanparams.pos+delanparams.seginfo[delanparams.seg*10]+delanparams.seginfo[delanparams.seg*10+1]+2)
    {
        delanparams.pos=posx-1+delanparams.pos;
```

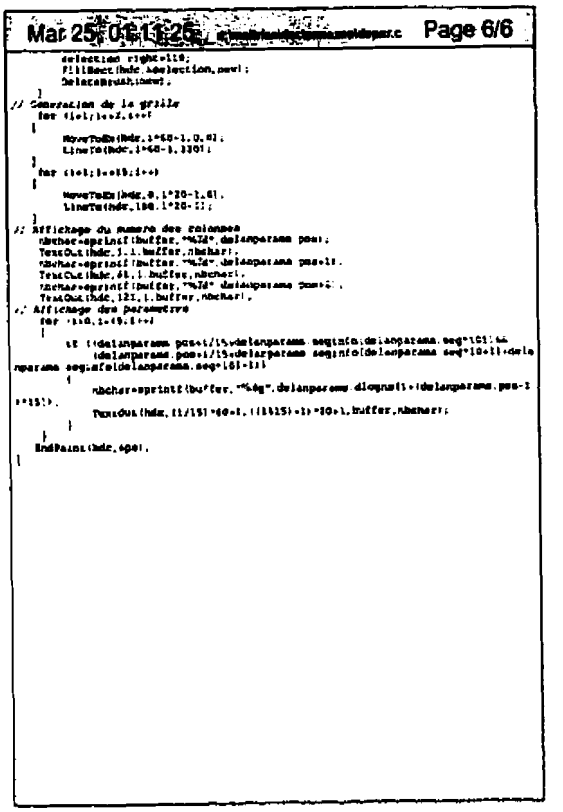


```
        posx=1;
    }
// On reaffiche tout
    InvalidateRect(hWnd,0,1);
}

//////////////////////////////////////////
// FONCTION     delan_params_killfocus
// DESCRIPTION  Appelee lorsque la fenetre perd l'avant plan
// PARAMETRES   hWnd: handler de la fenetre parametres
//              newHWnd: nouvelle fenetre a l'avant plan
// RETOUR       Aucun
//////////////////////////////////////////

void delan_params_killfocus(HWND hWnd,HWND newHWnd)
{
// Reaffichage de la fenetre
    InvalidateRect(hWnd,0,1);
}

//////////////////////////////////////////
// FONCTION     delan_params_command
// DESCRIPTION  Gestion des commandes du clavier
// PARAMETRES   hWnd: handler de la fenetre parametres
//              wID: handler de la fenetre enfant
//              hwndCtl: handler du control appelant
//              wNotifyCode: type de message
// RETOUR       Aucun
//////////////////////////////////////////

void delan_params_command(HWND hWnd,int wID,HWND hwndCtl,int wNotifyCode)
{

}

//////////////////////////////////////////
// FONCTION     delan_params_keydown
// DESCRIPTION  Gestion de la pression d'une touche de clavier
// PARAMETRES   hWnd: handler de la fenetre parametres
//              vKey: numero de la touche
//              tmp,x,y: inutilise
// RETOUR       Aucun
//////////////////////////////////////////

void delan_params_keydown(HWND hWnd,int vKey,int tmp,int x,int y)
{
    WNDPROC oldproc;
    char buffer[50];

// Deplacement d'une colonne vers la gauche
    if (vKey==VK_LEFT)
    {
        if (delanparams.pos>delanparams.seginfo[delanparams.seg*10]) delanparams.pos--;
    }
// Deplacement d'une colonne vers la droite
    if (vKey==VK_RIGHT)
    {
        if (delanparams.pos<delanparams.seginfo[delanparams.seg*10]+delanparams.seginfo[delanparams.seg*10+1]) delanparams.pos++;
    }
// Deplacement d'une case vers le haut
    if (vKey==VK_UP)
    {
        if (delanparams.curseur>0) delanparams.curseur--;
```



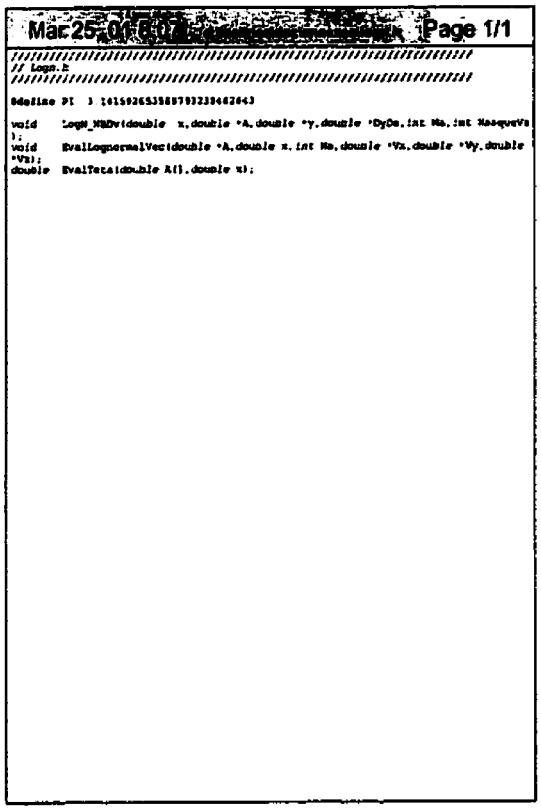
Mar 25, 01 11:25 Page 5/6

```
    }
// Deplacement d'une case vers le bas
    if (vkey==VK_DOWN)
    {
        if (delanparams.curseur<10) delanparams.curseur++;
    }
// Selection d'une case
    if (vkey==VK_RETURN&&delanparams.pos<delanparams.seqinfo[delanparams.seq*10]
+delanparams.seqinfo[delanparams.seq*10+1])
    {
// Si la fenetre d'edition existe on la detruit
        if (edit_win!=0) DestroyWindow(edit_win);
        if (delanparams.curseur<1)
        {
// Initialisation et creation de la fenetre d'edition
            edit_win=CreateWindow("EDIT",0,WS_CHILD|WS_VISIBLE|WS_BORDER,61,(del
anparams.curseur-1)*20+1,57,17,hWnd,(HMENU) 1451,hInst,0);
            SetFocus(edit_win);
            oldproc=SubclassWindow(edit_win,delan_child_edit);
            SetWindowLong(edit_win,GWL_USERDATA,(DWORD) oldproc);
            sprintf(buffer,"%.0g",delanparams.diognal[delanparams.pos*15+delanper
ams.curseur]);
            Edit_SetText(edit_win,buffer);
        }
        if (delanparams.curseur>=1) delanparams.diognal[delanparams.pos*15+1]=
delanparams.diognal[delanparams.pos*15+1];
    }
// On reaffiche tout
    InvalidateRect(hWnd,0,1);
}

////////////////////////////////////////
// FONCTION      delan_param_paint
// DESCRIPTION   Fonction d'affichage de la fenetre parametres
// PARAMETRES    HWnd: handler de la fenetre parametres
// RETOUR        Aucun
////////////////////////////////////////

void delan_param_paint(HWND hWnd)
{
    PAINTSTRUCT ps;
    HDC hdc;
    HBRUSH new;
    int i,nbchar;
    char buffer[25];
    RECT selection;

    hdc=BeginPaint(hWnd,&ps);
// Si nous sommes en premier plan
    if (GetFocus()==hWnd||GetFocus()==edit_win)
    {
// Affichage de la colonne courante
        new=CreateSolidBrush(RGB(238,255,6));
        SetMapMode(hdc,TRANSPARENT);
        selection.top=0;
        selection.bottom=18;
        selection.left=60;
        selection.right=19;
        FillRect(hdc,&selection,new);
        DeleteBrush(new);
// Affichage du curseur
        new=CreateSolidBrush(RGB(0,255,0));
        selection.top=20+delanparams.curseur*20;
        selection.bottom=38+delanparams.curseur*20;
        selection.left=60;
```



Mar 25, 01 11:25 Page 6/6

```
        selection.right=119;
        FillRect(hdc,&selection,new);
        DeleteBrush(new);
    }
// Generation de la grille
    for (i=1;i<=2;i++)
    {
        MoveToEx(hdc,i*60-1,0,0);
        LineTo(hdc,i*60-1,330);
    }
    for (i=1;i<=19;i++)
    {
        MoveToEx(hdc,0,i*20-1,0);
        LineTo(hdc,180,i*20-1);
    }
// Affichage du numero des colonnes
    nbchar=sprintf(buffer,"%3d",delanparams.pos);
    TextOut(hdc,1,1,buffer,nbchar);
    nbchar=sprintf(buffer,"%3d",delanparams.pos+1);
    TextOut(hdc,61,1,buffer,nbchar);
    nbchar=sprintf(buffer,"%3d",delanparams.pos+2);
    TextOut(hdc,121,1,buffer,nbchar);
// Affichage des parametres
    for (i=0,i<=9,i++)
    {
        if ((delanparams.pos+i)<=delanparams.seqinfo[delanparams.seq*10]&&
            (delanparams.pos+i)<=delanparams.seqinfo[delanparams.seq*10+1]+dela
nparams.seqinfo[delanparams.seq*10]-1))
        {
            nbchar=sprintf(buffer,"%.0g",delanparams.diognal[i+(delanparams.pos-1
)*15]);
            TextOut(hdc,(1/15)*60+1,((1515)+1)*20+1,buffer,nbchar);
        }
    }
    EndPaint(hdc,&ps);
}
```



Mar25 01 6:07 Page 1/1

```
////////////////////////////////////////////////////////////////
// Logn.h
////////////////////////////////////////////////////////////////

#define PI  3.14159265358979323846264

void    LogN_NSDrv(double  x,double *A,double *y,double *DyDx,int Na,int NsequeVa
);
void    EvalLognormalVec(double *A,double x,int Na,double *Vx,double *Vy,double
*Vz);
double  EvalTeta(double A[],double x);
```

Mar 25, 01 11:58 d:\malbrieu\doctorat\sauvegarde\logn.c Page 1/5

```
//////////////////////////////////////
// PROGRAMME     Calcul des delta-lognormales
// BUT           Calculer la vitesse v l'aide de la
//               fonction delta-lognormale
// FICHIER       logn.C
// DESCRIPTION   Permet de calculer la vitesse en X,Y et Z de
//               trajectoires decrites par le modele delta-lognormal
//               On peut aussi calculer les derivee partielles
// REMARQUE      Aucune
// AUTEUR        Nicolas Leduc et Wacef Guerfali
// DATE          15 septembre 2000
//////////////////////////////////////

#include <math.h>
#include <stdio.h>
#include "logn.h"
#include "nrutil.h"
#include "vecnrd.h"

//////////////////////////////////////////////////////////////////////
//
// ATTENTION:
// ==========
// LES INDICES DE TOUS LES VECTEURS VARIENT DE 1 a N ET NON DE 0 a N-1
// AFIN D'EVITER LES CONFUSIONS DUES A L'UTILISATION DES FONCTIONS DE
// NUMERICAL RECIPES IN C
//
//////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////
// Fonction d'evaluation des derivees partielles autour d'un point
// pour la fonction Delta Lognormale (double) Vectoriel
// La fonction recoit comme argument:
// x: la valeur de l'abcisse ou on veut evaluer les derivees
// A: le vecteur de parametres ou:
//    A[1] = D1
//    A[2] = Mu1
//    A[3] = Sigma1
//    A[4] = t0
//    A[5] = D2
//    A[6] = Mu2
//    A[7] = Sigma2
//    A[8] = C0
//    A[9] = Teta0
//    A[10]= Rotation
//    A[11]= Elevation
//    A[12]=
//    A[13]=
//    A[14]=
//    A[15]= On/Off
// y:    la valeur exacte de l'ordonne etant donne les parametres
// DyDa: Vecteur des derivees partielles par rapport aux 11 parametres
// Ma:   Nombre de parametres (15 dans ce cas)
//////////////////////////////////////////////////////////////////////

void    LogN_NRDv(double  x,        // Valeur de l'abcisse
                  double *A,        // Estimation des parametres
                  double *y,        // Pointeur valeur de l'ordonne
                  double *DyDa,     // Vecteur des derivees partielles
                  int    Ma,        // Nombre de parametres
                  int *masqueVa     // Masque des parametres a estimer
                 )
{
  int i,j,masque;
```

Mar 25, 01 11:58 d:\malbrieu\doctorat\sauvegarde\logn.c Page 2/5

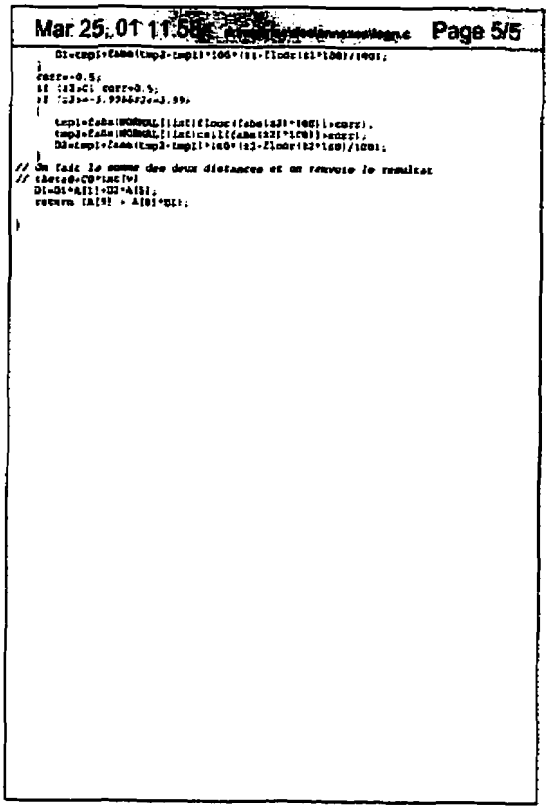
```
  double yp,ym,h,Teta;

  *y = 0.0;
// Evaluer la fonction resultante au point x
  EvalLognormalVect(A,x,Ma,y,&Teta);
// Pour chaque delta-Lognormale (groupe de 15 parametres)
  for (i=0;i<Ma;i+=15)
     {
       // Si x et plus petit que t0 alors y et toutes les derivee ne
       // sont pas calculable (retourner des 0)
       if (x <= A[i+4])
         {
          DyDa[i+1]=DyDa[i+2]=DyDa[i+3]=DyDa[i+4]=0.0;
          DyDa[i+5]=DyDa[i+6]=DyDa[i+7]=DyDa[i+8]=0.0;
          DyDa[i+9]=DyDa[i+10]=DyDa[i+11]=0.0;
         }
       else
         {
// Calculer les derivee partielles autour de x
// par differenciation ce cause du temps de calcul important
// Le masque indique les variables pour lesquelle on veut faire
//    le calcul
       masque = 1;
       // Pas assez petit
         h = 0.01;
       // Pour chaqu'un des neuf parametres
       for (j=1; j<=15; j++)
          {
           DyDa[i+j] = 0.0;
           // Si le calcul de la derive partielle est demande
              if (MasqueVa & masque)
                {
                  // Ajouter un pas de h au parametre A[i+j]
                  A[i+j] += h;
                // Evaluer la fonction en ce point
                  EvalLognormalVec(A,x,Ma,&yp,&Teta);
                // Enlever un pas de h au parametre A[i+j]
                  A[i+j] -= 3.5*h;
                // Evaluer la fonction en ce point
                  EvalLognormalVec(A,x,Ma,&ym,&Teta);
                // Retablir l'ancienne valeur du parametre A[i+j]
                  A[i+j] += h;
                // Estimer la derivee par la difference des resultats
                // divise par l'intervalle (soit 2 fois le pas)
                  DyDa[i+j] = (yp-ym)/(2 *h);
                }
           masque *= 2;
          }
       }
  }

//////////////////////////////////////////////////////////////////////
// Fonction d'evaluation d'une sommation vectorielle d'une fonction
// delta-lognormale.
// La fonction recoit comme argument.
// x: la valeur de l'abcisse ou on veut evaluer la fonction
// A: le vecteur de parametres ou:
//    A[1] = D1
//    A[2] = Mu1
//    A[3] = Sigma1
//    A[4] = t0
//    A[5] = D2
//    A[6] = Mu2
//    A[7] = Sigma2
```

Mar 25, 01 11:58 d:\malbrieu\doctorat\sauvegarde\logn.c Page 3/5

```
//    A[8] = C0
//    A[9] = Teta0
//    A[10]= Rotation
//    A[11]= Elevation
//    A[12]=
//    A[13]=
//    A[14]=
//    A[15]= On/Off
// Ma:   Nombre de parametres (multiple de 15 dans ce cas)
// Vit:  La valeur exacte de l'ordonne etant donne les parametres
// Ang:  La valeur exacte de l'angle etant donne les parametres
// ATTENTION: CETTE FONCTION TRAITE UN MAXIMUM DE 10 DELTA LOGNORMALES
//////////////////////////////////////////////////////////////////////
//#define MAX 100
void   EvalLognormalVec(double *A,      // Estimation des parametres
                        double x,       // Valeur a l'ordonne
                        int Ma,         // Nombre de parametres
                        double *Va,     // Vecteur vitesse en X
                        double *Vy,     // Vecteur vitesse en Y
                        double *Vt      // Vecteur vitesse en Z
                       )
  {
  int i;
  double Vxnew,Vynew,Vznew,Vxold,Vyold,Vzold;
  double Y,Theta,Y,E,Zn,Zp,SQ2PI;

  *Vx=0;
  *Vy=0;
  *Vz=0;
  for (i=0;i<Ma/15;i++)
  {
// Si on doit inclure la courbe dans le calcul
    if ((A[i*15+1] == 0.0) || (x <= A[i*15+4]) || (A[i*15+15] == 0));
    else
    {
      Y   = x - A[i*15+4];  // calcul de t-t0
      Z   = log(Y);         // calcul de log(t-t0)
      Zp  = Z - A[i*15+2];  // log(t-t0)-Mu1
      Zn  = Z + A[i*15+6];  // log(t-t0)-Mu2
      SQ2PI = sqrt(2*PI);
// Cas log-normale simple
      if (fabs(Zp)>0.00001)
      V = A[i*15+1] / (SQ2PI*A[i*15+3]*Y) *
          exp(-0.5/(A[i*15+3]*A[i*15+3])*Zp*Zp);
// Cas log-normale double
      if (fabs(Zn)>0.00001)
      V += A[i*15+5] / (SQ2PI*A[i*15+7]*Y) *
          exp(-0.5/(A[i*15+7]*A[i*15+7])*Zn*Zn);
// Evaluation de l'angle au point x pour la delta lognormale en cours
      Theta = EvalTeta(&A[i*15],x);
// Calcul du vecteur vitesse initial
      Vxold=V*cos(Theta);
      Vyold=V*sin(Theta);
      Vzold=0;
// Rotation autour de l'axe des X
      Vxnew=Vxold;
      Vynew=Vyold*cos(A[i*15+11])-Vzold*sin(A[i*15+11]);
      Vznew=Vyold*sin(A[i*15+11])+Vzold*cos(A[i*15+11]);
      Vxold=Vxnew;
      Vyold=Vynew;
      Vzold=Vznew;
// Rotation autour de l'axe des Z
      Vxnew=Vxold*cos(A[i*15+10])-Vyold*sin(A[i*15+10]);
      Vynew=Vxold*sin(A[i*15+10])+Vyold*cos(A[i*15+10]);
      Vznew=Vzold;
```

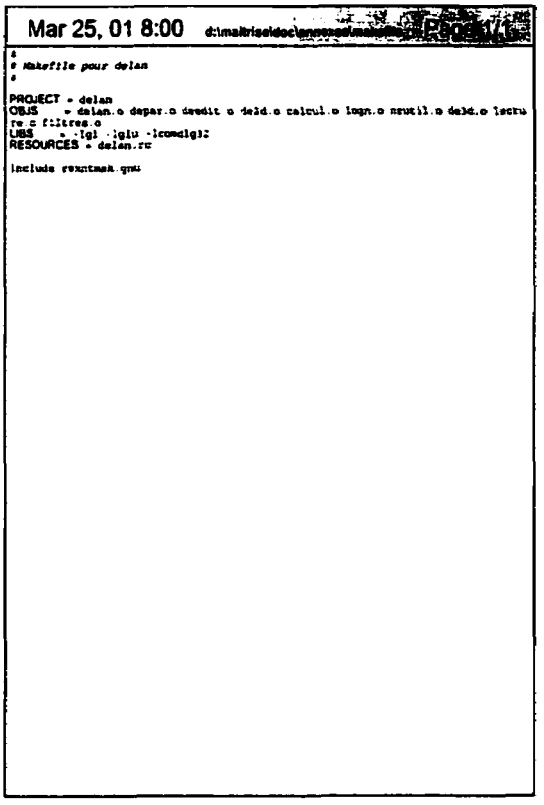
Mar 25, 01 11:58 d:\malbrieu\doctorat\sauvegarde\logn.c Page 4/5

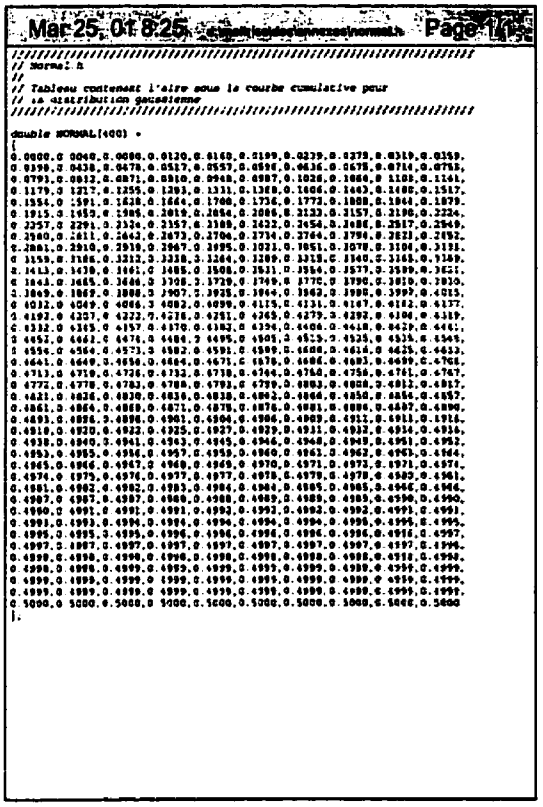
```
// Calcul de la sommation vectorielle
      *Vx+=Vxnew;
      *Vy+=Vynew;
      *Vz+=Vznew;
    }
  }
}

//////////////////////////////////////////////////////////////////////
// Fonction d'evaluation de l'angle d'une seule courbe
// delta-lognormale vectorielle.
// L'angle en un point donne est calcule par la fonction suivante:
//       Teta(t) = Teta0 + C0* Integrale(v(t)) entre t0 et t
// La fonction recoit comme argument.
// x: la valeur de l'abcisse ou on veut evaluer l'angle t(x)
// A: le vecteur de parametres ou.
//    A[1] = D1
//    A[2] = Mu1
//    A[3] = Sigma1
//    A[4] = t0
//    A[5] = D2
//    A[6] = Mu2
//    A[7] = Sigma2
//    A[8] = C0
//    A[9] = Teta0
//    A[10]= Rotation
//    A[11]= Elevation
//    A[12]=
//    A[13]=
//    A[14]=
//    A[15]= On/Off
// La fonction retourne comme valeur de retour l'angle au point x
//
// ATTENTION x DOIT ETRE PLUS GRAND QUE t0
//////////////////////////////////////////////////////////////////////

double  EvalTeta(double A[],      // Valeur des 9 parametres
                 double x         // Valeur de l'abcisse
                )
{
  double z1,z2,Z1,D2,x2,corr,tmp1,tmp2;//tmp3,tmp4;

// On calcule la cote z pour les deux log-normales qui forment
// la delta log-normale.  z=(ln(x)-mu)/sigma
  D1=0;
  D2=0;
  x2=0.00001;
  if (x-A[4]>0) x2=x-A[4];
  z1=-5;
  if (A[3]!=0) z1=(log(x2)-A[2])/A[3];
  z2=-5;
  if (A[7]!=0) z2=(log(x2)-A[6])/A[7];
  if (z1<-3.99) D1=0;
  if (z1>3.99) D1=1.
  if (z2<-3.99) D2=0.
  if (z2>3.99) D2=1;
  corr=-0.5;
  if (z1>0) corr=0.5;
// On lis dans une table l'air sous la courbe de - infini jusqu'a z
// On a donc le pourcentage de distance parcouru et on connait
// l'int(v) de t0 a t. On fait le meme calcul pour les deux log-normales
  if (z1>=-3.95&&z1<=3.99)
  {
    tmp1=fabs(NORMAL[(int)(floor(fabs(z1)*100))]+corr);
    tmp2=fabs(NORMAL[(int)(ceil(fabs(z1)*100))]+corr);
```

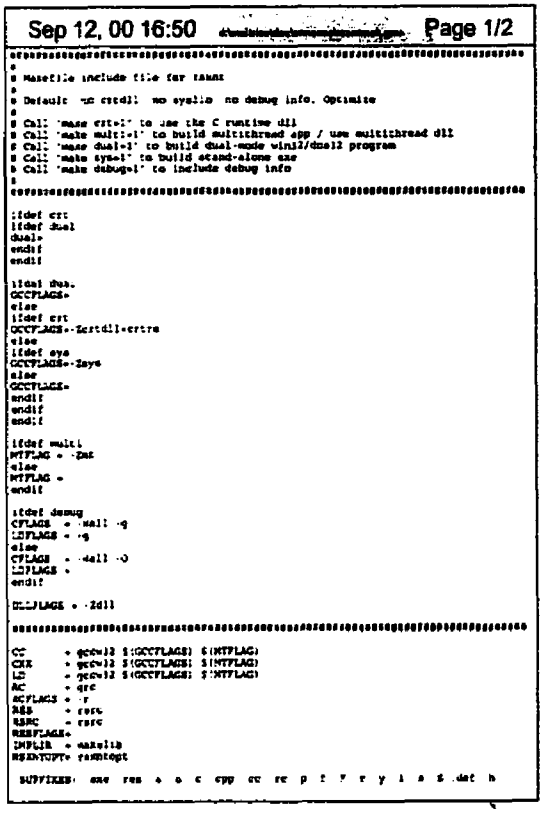


Mar 25,01 11:58 [illegible] Page 5/5

```
    D1=tmp1+fabs(tmp3-tmp1*106*(i1-Ilocr[i1*100])/100);
   }
   corr=-0.5;
   if (i3>C) corr=0.5;
   if (i3>=-3.9344372e+3.99)
   {
      tmp1=fabs(NORMAL[(int)(Ilocr[fabs(i3*100)]+corr).
      tmp1=fabs(NORMAL[(int)(a.1[(fabs(i2[*100)]+corr).
      D2=tmp1+fabs(tmp3-tmp1[*100*(i2-Ilocr[i2*100])/100);
   }
// On fait la somme des deux distances et on renvoie le resultat
// (Bersek+C0*inc[0]
   D1=D1*A[1]+D2*A[0];
   return (A[0] + A[0]*D1);

}
```

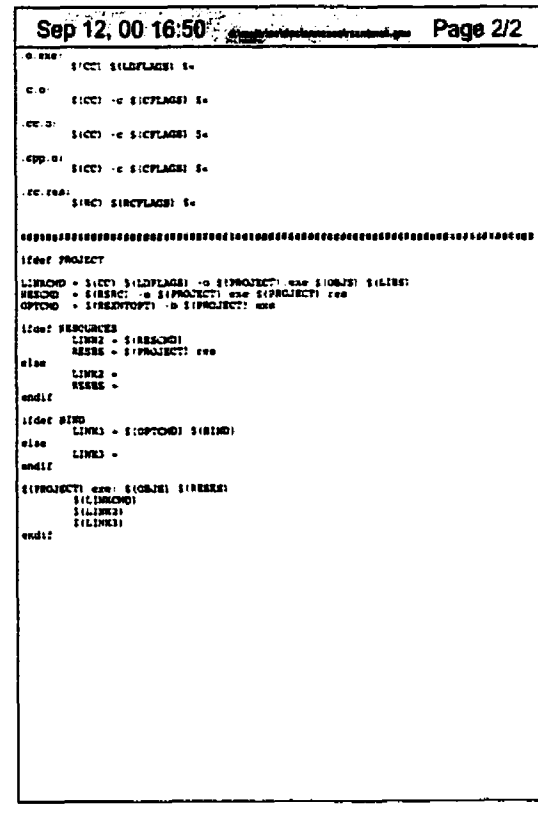

Mar 25, 01 8:00
Page 1/1
#
Makefile pour delan
#
PROJECT = delan
OBJS = delan.o depar.o dredit.o deld.o calcul.o logn.o nrutil.o deld.o lectu
re.o filtres.o
LIBS = -lgl -lglu -lcomdlg32
RESOURCES = delan.rc
include rsxntmak.gnu


Mar 25, 01 8:25
Page 1/1
//
// Normal.h
//
// Tableau contenant l'aire sous la courbe cumulative pour
// la distribution gaussienne
//
double NORMAL[400] =
{
0.0000,0.0040,0.0080,0.0120,0.0160,0.0199,0.0239,0.0279,0.0319,0.0359,
0.0398,0.0438,0.0478,0.0517,0.0557,0.0596,0.0636,0.0675,0.0714,0.0753,
0.0793,0.0832,0.0871,0.0910,0.0948,0.0987,0.1026,0.1064,0.1103,0.1141,
0.1179,0.1217,0.1255,0.1293,0.1331,0.1368,0.1406,0.1443,0.1480,0.1517,
0.1554,0.1591,0.1628,0.1664,0.1700,0.1736,0.1772,0.1808,0.1844,0.1879,
0.1915,0.1950,0.1985,0.2019,0.2054,0.2088,0.2123,0.2157,0.2190,0.2224,
0.2257,0.2291,0.2324,0.2357,0.2389,0.2422,0.2454,0.2486,0.2517,0.2549,
0.2580,0.2611,0.2642,0.2673,0.2704,0.2734,0.2764,0.2794,0.2823,0.2852,
0.2881,0.2910,0.2939,0.2967,0.2995,0.3023,0.3051,0.3078,0.3106,0.3133,
0.3159,0.3186,0.3212,0.3238,0.3264,0.3289,0.3315,0.3340,0.3365,0.3389,
0.3413,0.3438,0.3461,0.3485,0.3508,0.3531,0.3554,0.3577,0.3599,0.3621,
0.3643,0.3665,0.3686,0.3708,0.3729,0.3749,0.3770,0.3790,0.3810,0.3830,
0.3849,0.3869,0.3888,0.3907,0.3925,0.3944,0.3962,0.3980,0.3997,0.4015,
0.4032,0.4049,0.4066,0.4082,0.4099,0.4115,0.4131,0.4147,0.4162,0.4177,
0.4192,0.4207,0.4222,0.4236,0.4251,0.4265,0.4279,0.4292,0.4306,0.4319,
0.4332,0.4345,0.4357,0.4370,0.4382,0.4394,0.4406,0.4418,0.4429,0.4441,
0.4452,0.4463,0.4474,0.4484,0.4495,0.4505,0.4515,0.4525,0.4535,0.4545,
0.4554,0.4564,0.4573,0.4582,0.4591,0.4599,0.4608,0.4616,0.4625,0.4633,
0.4641,0.4649,0.4656,0.4664,0.4671,0.4678,0.4686,0.4693,0.4699,0.4706,
0.4713,0.4719,0.4726,0.4732,0.4738,0.4744,0.4750,0.4756,0.4761,0.4767,
0.4772,0.4778,0.4783,0.4788,0.4793,0.4798,0.4803,0.4808,0.4812,0.4817,
0.4821,0.4826,0.4830,0.4834,0.4838,0.4842,0.4846,0.4850,0.4854,0.4857,
0.4861,0.4864,0.4868,0.4871,0.4875,0.4878,0.4881,0.4884,0.4887,0.4890,
0.4893,0.4896,0.4898,0.4901,0.4904,0.4906,0.4909,0.4911,0.4913,0.4916,
0.4918,0.4920,0.4922,0.4925,0.4927,0.4929,0.4931,0.4932,0.4934,0.4936,
0.4938,0.4940,0.4941,0.4943,0.4945,0.4946,0.4948,0.4949,0.4951,0.4952,
0.4953,0.4955,0.4956,0.4957,0.4959,0.4960,0.4961,0.4962,0.4963,0.4964,
0.4965,0.4966,0.4967,0.4968,0.4969,0.4970,0.4971,0.4972,0.4973,0.4974,
0.4974,0.4975,0.4976,0.4977,0.4977,0.4978,0.4979,0.4979,0.4980,0.4981,
0.4981,0.4982,0.4982,0.4983,0.4984,0.4984,0.4985,0.4985,0.4986,0.4986,
0.4987,0.4987,0.4987,0.4988,0.4988,0.4989,0.4989,0.4989,0.4990,0.4990,
0.4990,0.4991,0.4991,0.4991,0.4992,0.4992,0.4992,0.4992,0.4993,0.4993,
0.4993,0.4993,0.4994,0.4994,0.4994,0.4994,0.4994,0.4995,0.4995,0.4995,
0.4995,0.4995,0.4995,0.4996,0.4996,0.4996,0.4996,0.4996,0.4996,0.4997,
0.4997,0.4997,0.4997,0.4997,0.4997,0.4997,0.4997,0.4997,0.4997,0.4998,
0.4998,0.4998,0.4998,0.4998,0.4998,0.4998,0.4998,0.4998,0.4998,0.4998,
0.4998,0.4998,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,
0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,
0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,0.4999,
0.5000,0.5000,0.5000,0.5000,0.5000,0.5000,0.5000,0.5000,0.5000,0.5000
};



Sep 12, 00 16:50 [redacted] Page 1/2

```
#
# Makefile include file for tamas
#
# Default -mt cvsdll  no symlib  no debug info. Optimize
#
# Call 'make crt=1' to use the C runtime dll
# Call 'make multi=1' to build multithread app / use multithread dll
# Call 'make dual=1' to build dual-mode win32/dos32 program
# Call 'make sys=1' to build stand-alone exe
# Call 'make debug=1' to include debug info
#

!ifdef crt
!ifdef dual
dual=
!endif
!endif

!ifdef dual
GCCFLAGS=
!else
!ifdef crt
GCCFLAGS=-lcrtdll=crtrs
!else
!ifdef sys
GCCFLAGS=-lsys
!else
GCCFLAGS=
!endif
!endif
!endif

!ifdef multi
MTFLAG = -Zmt
!else
MTFLAG =
!endif

!ifdef debug
CFLAGS  = -Wall -g
LDFLAGS = -g
!else
CFLAGS  = -Wall -O
LDFLAGS =
!endif

DLLFLAGS = -Zdll

CC      = gcc-32 $(GCCFLAGS) $(MTFLAG)
CPP     = gcc-32 $(GCCFLAGS) $(MTFLAG)
LD      = gcc-32 $(GCCFLAGS) $(MTFLAG)
AR      = grc
ARFLAGS = -r
RES     = rsrc
RSRC    = rsrc
RESFLAGS=
IMPLIB  = emximp
RSXNTOPT= rsxntopt

.SUFFIXES: .exe .res .a .o .c .cpp .rc .re .p .f .r .y .l .s .$$$ .def .h
```



Sep 12, 00 16:50 [redacted] Page 2/2

```
.o.exe:
        $(CC) $(LDFLAGS) $<

.c.o:
        $(CC) -c $(CFLAGS) $<

.cc.o:
        $(CC) -c $(CFLAGS) $<

.cpp.o:
        $(CC) -c $(CFLAGS) $<

.rc.res:
        $(RC) $(RCFLAGS) $<

#
# Rules for building the .exe file.
#
# Usage: make [debug=1] [dual=1] [crt=1] [sys=1] [mt=1] [cvs=1] [wrap=1] [nodos=1]
#

!ifdef PROJECT

LINKCMD = $(CC) $(LDFLAGS) -o $(PROJECT).exe $(OBJS) $(LIBS)
RESCMD  = $(RSRC) -o $(PROJECT).exe $(PROJECT).res
OPTCMD  = $(RSXNTOPT) -b $(PROJECT).exe

!ifdef $(RESOURCES)
        LINK2 = $(RESCMD)
        RESRS = $(PROJECT).res
else
        LINK2 =
        RESRS =
endif

!ifdef $(BIND)
        LINK3 = $(OPTCMD) $(BIND)
else
        LINK3 =
endif

$(PROJECT).exe: $(OBJS) $(RESRS)
        $(LINKCMD)
        $(LINK2)
        $(LINK3)
endif
```

Annexe 7

Numerical Receipes in C

Cette annexe présente les routines issues de Numerical Receipes in C qui ont été utilisés dans nos différents programmes. Ces routines sont écrites en C et utilisent des vecteurs commençant à l'indice 1 et non 0 comme nous les utilisons habituellement. Elles permettent de créer et de détruire des vecteurs dynamiquement. De plus elles contiennent un filtre dérivatif et un filtre passe-bas. Des utilitaires pour la lecture des fichiers sont aussi présentés.

```
////////////////////////////////////////////////////////////////////
// Filtres.h
////////////////////////////////////////////////////////////////////

double      *Filtre_pb_16(double *,int);
double      *Filtre_derivatif(double *,int);
double      *PpeTR(double *,double *,int,double *,int);
double      *FiltreTR(double *,double *,int,double *,int);
double      *FiltreFIR(double *,int,double *,int);
double      *FiltreIIR(double *,double *,int,double *,int);
double      *FiltreIIRCl(double *,double *,double *,int,double *,int);
double      *ReflatCX(double *,int,int);
double      *ReflatXD(double *,int,int);
double      *ReflatCXD(double *,int,int);
double      *SousVect(double *,int,int,int);
int         *IntSousVect(int *,int,int,int);
float       *CvtVdVf(double *,int);
double      *CvtVfVd(float  *,int);
```

```
////////////////////////////////////////////////////////////////////////
// Fonctions: Filtres.cpp
//
// Date de creation:   16 avril 1998
////////////////////////////////////////////////////////////////////////

#include "fdsouh"
#include "vmalh"
#include "vmlh"

////////////////////////////////////////////////////////////////////////
//
// ATTENTION:
// ==========
// LES INDICES DE TOUS LES VECTEURS VARIENT DE 1 a N ET NON DE 0 a N-1
// AFIN D'EVITER LES CONFUSIONS DANS L'UTILISATION DES FONCTIONS DE
// NUMERICAL RECIPES IN C
//
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Filtrer un signal par un passe bas ..................... filtre_pb_16
// Filtre: type = chebychev, ordre2 ordre.attenuation = 40dB
// Frequence de coupure =16Hz. N=13 coefficience.
// Les echantillons sont supposes etre a intervalle de 10ms (100Hz)
// La fonction recoit un pointeur sur le vecteur de points a filtrer
// ainsi que le nombre de point du vecteur.
// La fonction ne modifie pas le vecteur de donnees originales
// La fonction alloue dynamiquement un vecteur de la meme taille que
// le vecteur de donnees brutes, et retourne un pointeur sur ce vecteur
// de donnees filtrees.
////////////////////////////////////////////////////////////////////////
double *Filtre_pb_16(double *y, int Ndata)
  {
  int Nab,Nd,Nr;
  double  *A,*B,
  double  *Dar,*Dart,*Dat;

  // Initialiser les coefficients du filtre
  // Voir Matlab pour le calcul des coefficients
  Nab = 13.
  A = dvector(1,Nab);
  B = dvector(1,Nab);
  A[1]  =  1.0000000000000e+00;
  A[2]  = -1.1051262309514e+00,
  A[3]  =  6.1616464616760e+00;
  A[4]  = -7.9167573705322e+00,
  A[5]  =  6.8951874000884e+00,
  A[6]  = -4.5086489818151e+00,
  A[7]  =  2.0926590161543e+00;
  A[8]  = -0.2383881047161e+01,
  A[9]  =  3.3556929659868e-01;
  A[10] = -3.7192180092111e-02;
  A[11] =  8.3528489620863e-03;
  A[12] = -6.0856317068159e-04;

  B[1]  =  2.5735637256718e-03;
  B[2]  = -1.0325551022924e-02;
  B[3]  =  6.9931013113656e-02,
  B[4]  = -1.1775568984990e-02;
  B[5]  =  7.8644864914722e-02;
  B[6]  =  5.5892369918661e-02;
  B[7]  =  5.5882169258797e-03;
  B[8]  =  7.0544848218718e-02;
  B[9]  =  1.1775545988849e-02;
```



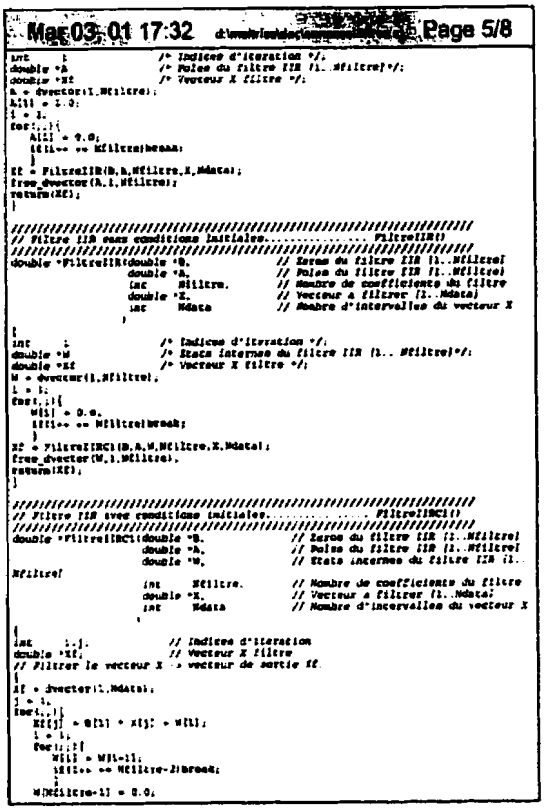
```
  B[10] =  6.9911013113656e-02,
  B[11] = -1.0325551022924e-02;
  B[12] =  2.5735637256718e-03;

  // Reflechir le vecteur y a gauche et a droite pour attenuer
  // l'effet des hautes frequences sur les bords du vecteur
  Nr = min(Ndata-1, 3*(Nab-1));
  Dar = RefletGXD(y,Ndata,Nr);
  Nd = Ndata + 2 * Nr;

  // Filtrer le vecteur y reflechi
  // avec la methode "Time Reversal"
  Dart = FiltreTR(B,A,Nab,Dar,Nd);

  // Eliminer les points reflechis.
  free_dvector(Dar,1,Nd);
  Dat = SousVect(Dart,Nd,Nr+1,Nr+Ndata);
  // Liberer les variables de travail
  free_dvector(Dart,1,Nd);
  free_dvector(A,1,Nab);
  free_dvector(B,1,Nab);

  return Dat;
  }

////////////////////////////////////////////////////////////////////////
// Calculer la derivee d'un signal ..................... filtre_derivee()
////////////////////////////////////////////////////////////////////////
double *Filtre_derivee(double *y, int Ndata)
  {
  int Nc,Nd,i,
  double *C,*Cbr,*Cdf,*Cbrfe;
  Nc  = 9.
  if(Ndata >= Nc){
//  Derivation par differenciation
    {
    Cbrfe = dvector(1,Ndata),
    i = 1;
     for(;;){
     Cbrfe[i] = y[i+1]   -y[i].
     if(++i >= Ndata-1)break;
     }
   Cbrfe[Ndata] = Cbrfe[Ndata-1];
}
   }
else{
  C = dvector(1,Nc);
  C[1]  = -4.5159572e-01,
  C[2]  =  1.8864795491801e+00,
  C[3]  = -2.2351521815936e-01,
  C[4]  =  4.2271592654621e-01,
  C[5]  =  0.0000000000000e+00,
  C[6]  = -4.2271592654621e-01,
  C[7]  =  2.2351521815936e-01,
  C[8]  = -1.8864795491801e+00,
  C[9]  =  4.5159572611957e-01;
  Cbrfe =dvector(1,Nc);
  Nd    = (Nc-1)/2,
  Cbr   = RefletGXD(y,Ndata,Nd),
  Cdf = FiltreFIR(C,Nc,Cbr,Ndata+2*Nd),
  Cbrfe= SousVect(Cdf,Ndata+2*Nc,Nc+Nd+1,Ndata+Nc+Nd),
  free_dvector(Cbr,1,Ndata+2*Nc),
  free_dvector(Cdf,1,Ndata+2*Nc),
  free_dvector(C,1,Nc),
  }
```



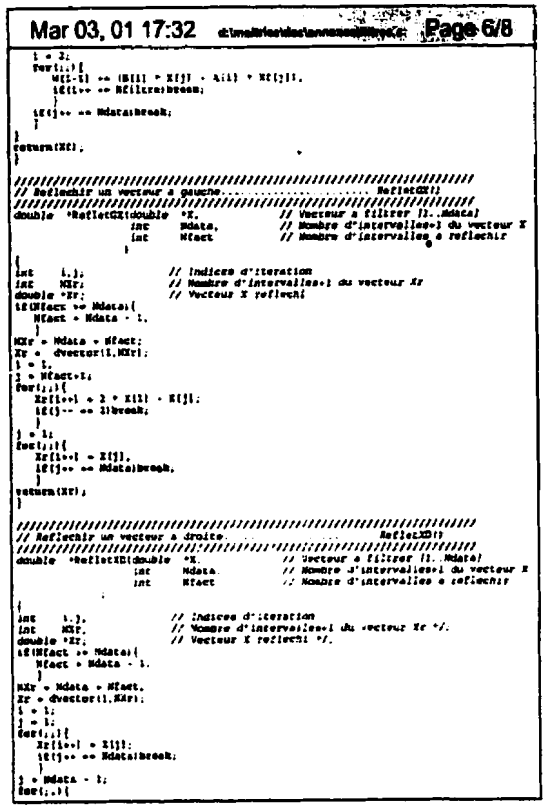
```
return(Cbrfe);
}

////////////////////////////////////////////////////////////////////////
// Filtre Passe-Bas avec la methode "Time Reversal" ..... FpbTR()
////////////////////////////////////////////////////////////////////////
double *FpbTR(double *B,          // Zeros du filtre IIR
              double *A,          // Poles du filtre IIR
              int     NFiltre,    // Nombre de coefficients du filtre
              double *X,          // Vecteur a filtrer [1..Ndata]
              int     Ndata       // Nombre d'intervalles du vecteur X
             )
  {
  int    NXr;              // Dimension du vecteur Xr
  int    Nfact;            // Dimension de la reflection
  double *Xr;              // Vecteur X reflechi
  double *Xrf;             // Vecteur Xr filtre
  double *XF;              // Vecteur X  filtre
  Nfact = min(Ndata-1,3*(NFiltre-1));
  Xr    = RefletGXD(X,Ndata,Nfact);
  NXr   = Ndata + 2 * Nfact;
  Xrf   = FiltreTR(B,A,NFiltre,Xr,NXr);
  XF    = SousVect(Xrf,Nfact,Nfact+Ndata,
  free_dvector(Xr,1,NXr);
  free_dvector(Xrf,1,NXr);
  return(XF);
  }

////////////////////////////////////////////////////////////////////////
// Filtre IIR avec la methode "Time Reversal" ..... FiltreTR()
////////////////////////////////////////////////////////////////////////
double *FiltreTR(double *B,          // Zeros du filtre IIR
                 double *A,          // Poles du filtre IIR
                     int     NFiltre,    // Nombre de coefficients du filtre
                     double *X,          // Vecteur a filtrer [1..Ndata]
                     int     Ndata       // Nombre d'intervalles du vecteur X
                    )
{
int     i,j;
int     NFact;
double *Zi;
double *Mm;
double *Xr;
double *Xrf;
int     NXr;
double *Y ;
int     NY ;
double *Yr;
double *Yrf;
double *Xf;
int     NYr;
// Initialiser les etats internes du filtre IIR.
{
if(A[1] != 1.){
   i = 2;
   for(;;){
      A[i] /= (A[1]);
      if(i++ >= NFiltre)break;
      }
   }
Zi = dvector(1,NFiltre);
Mm = dvector(1,NFiltre);
j = NFiltre;
i = 1.
for(;;){
```



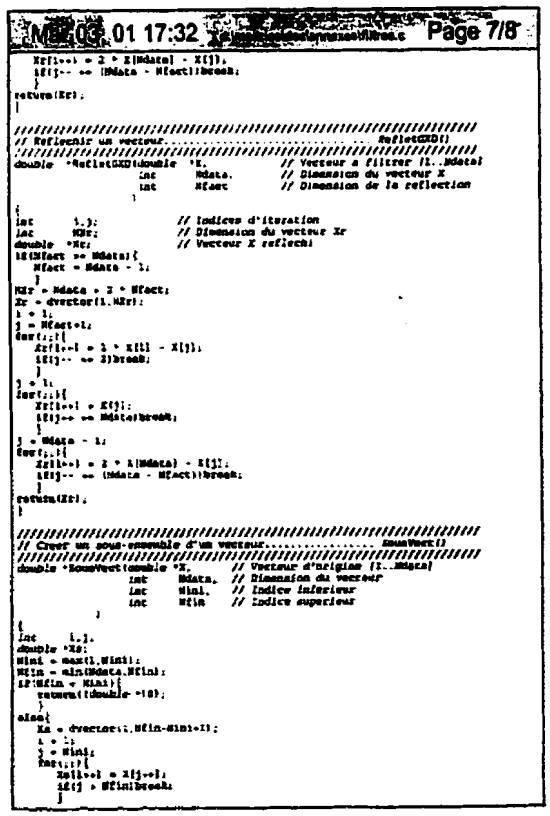
```
   Zi[i++] = (B[j]-A[j]);
   if(j-- == 1)break;
   }
i = 2;
for(;;){
   Zi[i] += Zi[i-1];
   if(i++ >= NFiltre)break;
   }
}
// Reflechir le vecteur a filtrer et le filtrer
{
NFact   =  3 * (NFiltre-1),
Xr      =  RefletGXD(X,Ndata,Nfact),
NXr     =  Ndata + NFact .
i = 1.
j = NFiltre-1.
for(;;){
   Mm[i++] = Zi[j] * Xr[1].
   if(j-- == 1)break;
   }
Xrf     =  FiltreIIRC1(B,A,Mm,NFiltre,Xr,NXr),
NY      =  NXr.
Y       =  dvector(1,NY);
i = 1;
j = NXr;
for(;;){
   Y[i++] = Xr[j];
   if(j-- == 1)break;
   }
Yr      =  RefletGXD(Y,NY,Nfact),
NYr     =  NY - Nfact .
i = 1;
j = NFiltre-1;
for(;;){
   Mm[i++] = Zi[j] * Yr[1];
   if(j-- == 1)break;
   }
Yrf     =  FiltreIIRC1(B,A,Mm,NFiltre,Yr,NYr),
Xf      =  dvector(1,Ndata);
i = 1;
j = NFact+Ndata;
for(;;){
   Xf[i] = Yrf[j--];
   if(i++ == Ndata)break;
   }
free_dvector(Zi ,1,NFiltre);
free_dvector(Mm ,1,NFiltre);
free_dvector(Xr  ,1,NXr);
free_dvector(Xrf ,1,NXr);
free_dvector(Y   ,1,NXr),
free_dvector(Yr  ,1,NYr);
free_dvector(Yrf ,1,NYr),
}
return(Xf);
}

////////////////////////////////////////////////////////////////////////
// Filtre FIR ..................................... FiltreFIR()
////////////////////////////////////////////////////////////////////////
double *FiltreFIR(double *B,          // Zeros du filtre IIR [1..NFiltre]
                  int     NFiltre,    // Nombre de coefficients du filtre
                  double *X,          // Vecteur a filtrer [1..Ndata]
                  int     Ndata       // Nombre d'intervalles du vecteur X
                 )
{
```



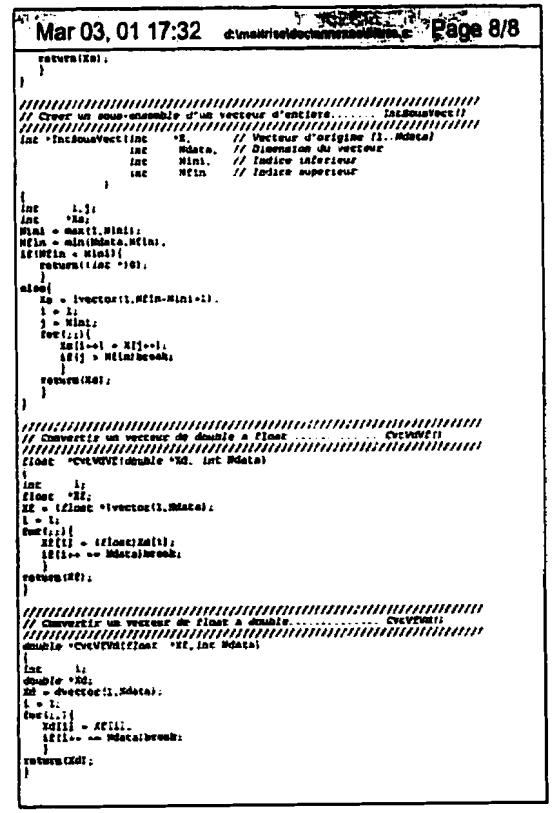
```
Mar 03, 01 17:32                                   Page 5/8
int     i,                      /* Indices d'iteration */;
double *A                       /* Poles du filtre IIR [1..Nfiltre]*/;
double *Xf                      /* Vecteur X filtre */;
A = dvector(1,Nfiltre);
A[1] = 1.0;
i = 1;
for(;;){
   A[i] = 0.0;
   if(i++ == Nfiltre)break;
   }
Xf = FiltreIIR(B,A,Nfiltre,X,Ndata);
free_dvector(A,1,Nfiltre);
return(Xf);
}

////////////////////////////////////////////////////////////////
// Filtre IIR sans conditions initiales.................. FiltreIIR()
////////////////////////////////////////////////////////////////
double *FiltreIIR(double *B,            // Zeros du filtre IIR [1..Nfiltre]
                  double *A,            // Poles du filtre IIR [1..Nfiltre]
                  int    Nfiltre,       // Nombre de coefficients du filtre
                  double *X,            // Vecteur a filtrer [1..Ndata]
                  int    Ndata         // Nombre d'intervalles du vecteur X
                 )
{
int     i;                      /* Indices d'iteration */,
double *W                       /* Etats internes du filtre IIR [1.. Nfiltre]*/;
double *Xf                      /* Vecteur X filtre */;
W = dvector(1,Nfiltre);
i = 1;
for(;;){
   W[i] = 0.0,
   if(i++ == Nfiltre)break;
   }
Xf = FiltreIIRC1(B,A,W,Nfiltre,X,Ndata);
free_dvector(W,1,Nfiltre),
return(Xf);
}

////////////////////////////////////////////////////////////////
// Filtre IIR avec conditions initiales.................. FiltreIIRC1()
////////////////////////////////////////////////////////////////
double *FiltreIIRC1(double *B,          // Zeros du filtre IIR [1..Nfiltre]
                    double *A,          // Poles du filtre IIR [1..Nfiltre]
                    double *W,          // Etats internes du filtre IIR [1..
Nfiltre]
                    int    Nfiltre,     // Nombre de coefficients du filtre
                    double *X,          // Vecteur a filtrer [1..Ndata]
                    int    Ndata        // Nombre d'intervalles du vecteur X
                   )
{
int     i,j,            // Indices d'iteration
double *Xf;             // Vecteur X filtre
// Filtrer le vecteur X -> vecteur de sortie Xf.
{
Xf = dvector(1,Ndata);
j = 1,
for(;;){
   Xf[j] = B[1] * X[j] + W[1];
   i = 1;
   for(;;){
      W[i] = W[i+1];
      if(i++ == Nfiltre-2)break;
      }
   W[Nfiltre-1] = 0.0;
```



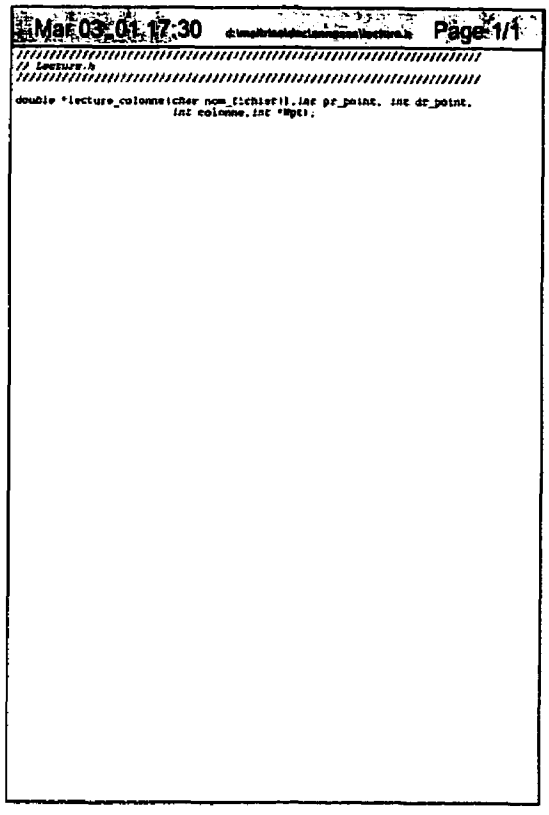
```
Mar 03, 01 17:32                                   Page 6/8
   i = 2;
   for(;;){
      W[i-1] += (B[i] * X[j] - A[i] * Xf[j]),
      if(i++ == Nfiltre)break;
      }
   if(j++ == Ndata)break;
   }
}
return(Xf),
}

////////////////////////////////////////////////////////////////
// Reflechir un vecteur a gauche.......................... RefletGX()
////////////////////////////////////////////////////////////////
double *RefletGX(double  *X,            // Vecteur a filtrer [1..Ndata]
                 int     Ndata,         // Nombre d'intervalles-1 du vecteur X
                 int     Nfact          // Nombre d'intervalles a reflechir
                )
{
int     i,j;            // Indices d'iteration
int     NXr;            // Nombre d'intervalles-1 du vecteur Xr
double *Xr;             // Vecteur X reflechi
if(Nfact >= Ndata){
   Nfact = Ndata - 1.
   }
NXr = Ndata + Nfact;
Xr = dvector(1,NXr);
i = 1;
j = Nfact+1;
for(;;){
   Xr[i++] = 2 * X[1] - X[j];
   if(j-- == 2)break;
   }
j = 1;
for(;;){
   Xr[i++] = X[j],
   if(j++ == Ndata)break,
   }
return(Xr);
}

////////////////////////////////////////////////////////////////
// Reflechir un vecteur a droite.......................... RefletXD()
////////////////////////////////////////////////////////////////
double  *RefletXD(double  *X,           // Vecteur a filtrer [1..Ndata]
                  int     Ndata,        // Nombre d'intervalles-1 du vecteur X
                  int     Nfact         // Nombre d'intervalles a reflechir
                 )
{
int     i,j,            // Indices d'iteration
int     NXr,            // Nombre d'intervalles-1 du vecteur Xr */.
double *Xr;             // Vecteur X reflechi */.
if(Nfact >= Ndata){
   Nfact = Ndata - 1.
   }
NXr = Ndata + Nfact,
Xr = dvector(1,NXr);
i = 1;
j = 1;
for(;;){
   Xr[i++] = X[j];
   if(j++ == Ndata)break;
   }
j = Ndata - 1;
for(;;){
```

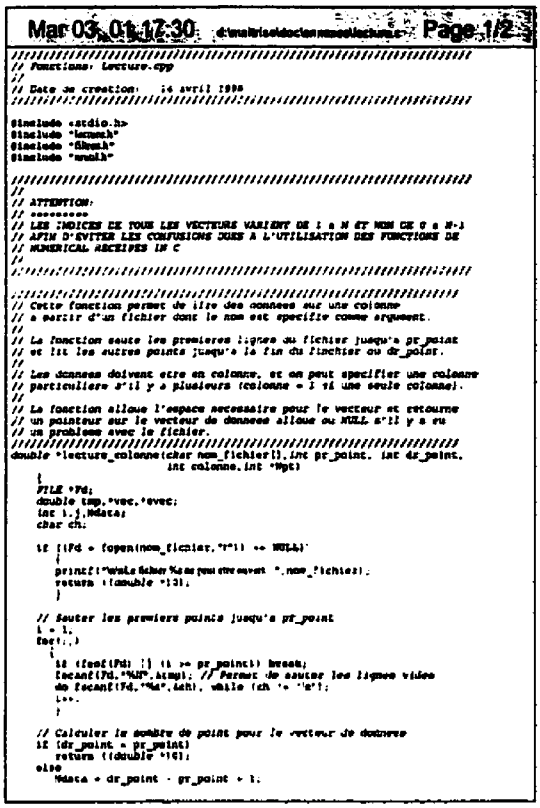


```
Mar 03, 01 17:32                                   Page 7/8
   Xr[i++] = 2 * X[Ndata] - X[j],
   if(j-- == (Ndata - Nfact))break;
   }
return(Xr);
}

////////////////////////////////////////////////////////////////
// Reflechir un vecteur................................... RefletGXD()
////////////////////////////////////////////////////////////////
double  *RefletGXD(double  *X,          // Vecteur a filtrer [1..Ndata]
                   int     Ndata,       // Dimension du vecteur X
                   int     Nfact        // Dimension de la reflection
                  )
{
int     i,j;            // Indices d'iteration
int     NXr;            // Dimension du vecteur Xr
double *Xr;             // Vecteur X reflechi
if(Nfact >= Ndata){
   Nfact = Ndata - 1.
   }
NXr = Ndata + 2 * Nfact;
Xr = dvector(1,NXr);
i = 1;
j = Nfact+1;
for(;;){
   Xr[i++] = 2 * X[1] - X[j];
   if(j-- == 2)break;
   }
j = 1;
for(;;){
   Xr[i++] = X[j];
   if(j++ == Ndata)break;
   }
j = Ndata - 1;
for(;;){
   Xr[i++] = 2 * X[Ndata] - X[j];
   if(j-- == (Ndata - Nfact))break;
   }
return(Xr);
}

////////////////////////////////////////////////////////////////
// Creer un sous-ensemble d'un vecteur.................... SousVect()
////////////////////////////////////////////////////////////////
double *SousVect(double *X,             // Vecteur d'origine [1..Ndata]
                 int     Ndata,         // Dimension du vecteur
                 int     Mini,          // Indice inferieur
                 int     Mfin           // Indice superieur
                )
{
int     i,j;
double *Xs;
Mini = max(1,Mini);
Mfin = min(Ndata,Mfin);
if(Mfin < Mini){
   return((double *)0);
   }
else{
   Xs = dvector(1,Mfin-Mini+1);
   i = 1;
   j = Mini;
   for(;;){
      Xs[i++] = X[j++];
      if(j > Mfin)break;
      }
   return(Xs);
   }
}
```

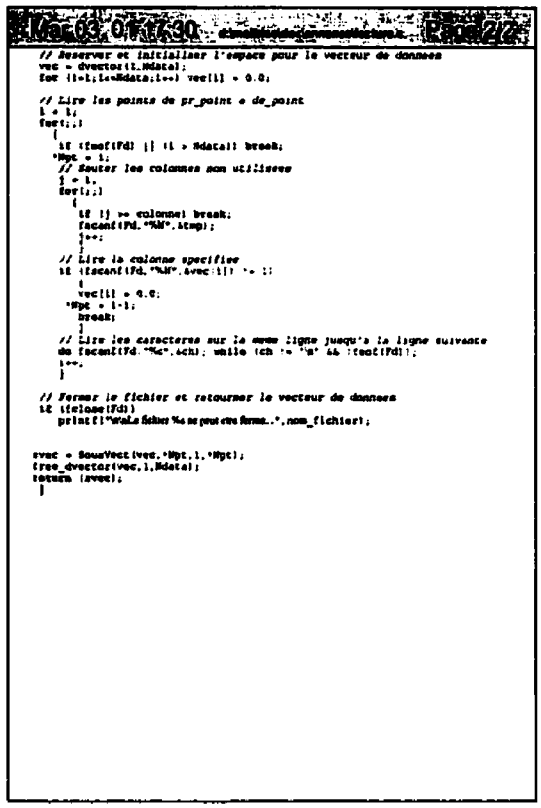


```
Mar 03, 01 17:32                                   Page 8/8
   return(Xs);
   }
}

////////////////////////////////////////////////////////////////
// Creer un sous-ensemble d'un vecteur d'entiers.......... IntSousVect()
////////////////////////////////////////////////////////////////
int *IntSousVect(int     *X,            // Vecteur d'origine [1..Ndata]
                 int     Ndata,         // Dimension du vecteur
                 int     Mini,          // Indice inferieur
                 int     Mfin           // Indice superieur
                )
{
int     i,j;
int     *Xs;
Mini = max(1,Mini);
Mfin = min(Ndata,Mfin);
if(Mfin < Mini){
   return((int *)0);
   }
else{
   Xs = ivector(1,Mfin-Mini+1);
   i = 1;
   j = Mini;
   for(;;){
      Xs[i++] = X[j++];
      if(j > Mfin)break;
      }
   return(Xs);
   }
}

////////////////////////////////////////////////////////////////
// Convertir un vecteur de double a float ................ CvtVdVf()
////////////////////////////////////////////////////////////////
float  *CvtVdVf(double *Xd, int Ndata)
{
int     i;
float  *Xf;
Xf = (float *)ivector(1,Ndata);
i = 1;
for(;;){
   Xf[i] = (float)Xd[i];
   if(i++ == Ndata)break;
   }
return(Xf);
}

////////////////////////////////////////////////////////////////
// Convertir un vecteur de float a double ................ CvtVfVd()
////////////////////////////////////////////////////////////////
double *CvtVfVd(float  *Xf, int Ndata)
{
int     i;
double *Xd;
Xd = dvector(1,Ndata);
i = 1;
for(;;){
   Xd[i] = Xf[i],
   if(i++ == Ndata)break;
   }
return(Xd);
}
```

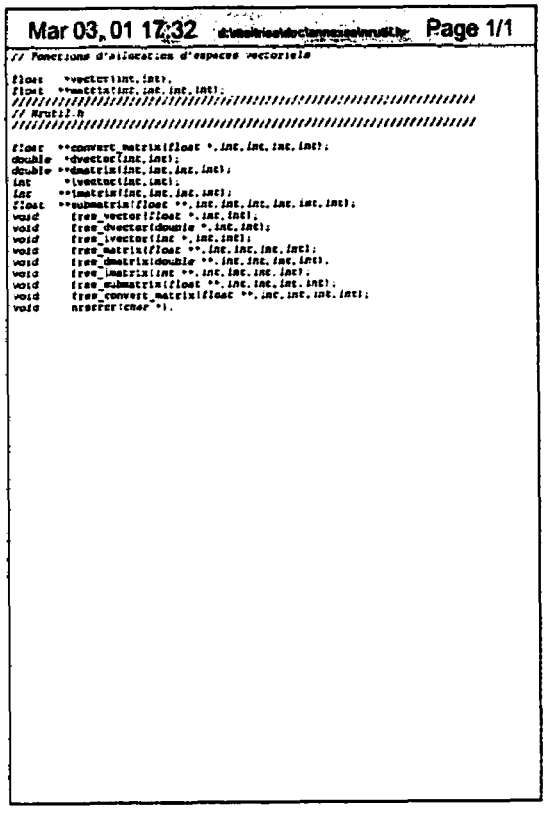

Mar 03_01 17:30
Page 1/1
// Lecture.h
double *lecture_colonne(char nom_fichier[],int pr_point, int dr_point,
int colonne,int *Mpt);



```
Mar 03, 01 12:30  d:\maitrise\doctorat\mesdocs\lecture.c  Page 1/2

////////////////////////////////////////////////////////////////////////
// Fonctions: Lecture.cpp
//
// Date de creation:   14 avril 1998
////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include "lecture.h"
#include "fbmat.h"
#include "vmath.h"

////////////////////////////////////////////////////////////////////////
//
// ATTENTION:
// ----------
// LES INDICES DE TOUS LES VECTEURS VARIENT DE 1 a N ET NON DE 0 a N-1
// AFIN D'EVITER LES CONFUSIONS DUES A L'UTILISATION DES FONCTIONS DE
// NUMERICAL RECIPES IN C
//
////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////
// Cette fonction permet de lire des donnees sur une colonne
// a partir d'un fichier dont le nom est specifie comme argument.
//
// La fonction saute les premieres lignes du fichier jusqu'a pr_point
// et lit les autres points jusqu'a la fin du fichier ou dr_point.
//
// Les donnees doivent etre en colonne, et on peut specifier une colonne
// particuliere s'il y a plusieurs (colonne = 1 si une seule colonne).
//
// La fonction alloue l'espace necessaire pour le vecteur et retourne
// un pointeur sur le vecteur de donnees alloue ou NULL s'il y a eu
// un problemes avec le fichier.
//////////////////////////////////////////////////////////////////////
double *lecture_colonne(char nom_fichier[],int pr_point, int dr_point,
                        int colonne,int *Npt)
    {
    FILE *Fd;
    double tmp,*vec,*vec;
    int i,j,Ndata;
    char ch;

    if ((Fd = fopen(nom_fichier,"r")) == NULL)
       {
       printf("\nLe fichier %s ne peut etre ouvert !",nom_fichier);
       return ((double *)0);
       }

    // Sauter les premiers points jusqu'a pr_point
    i = 1;
    for(;;)
      {
      if (feof(Fd) || (i >= pr_point)) break;
      fscanf(Fd,"%lf",&tmp); // Permet de sauter les lignes vides
      do fscanf(Fd,"%c",&ch); while (ch != '\n');
      i++;
      }

    // Calculer le nombre de point pour le vecteur de donnees
    if (dr_point < pr_point)
       return ((double *)0);
    else
       Ndata = dr_point - pr_point + 1;
```



```
Mar 03, 01 12:30  d:\maitrise\doctorat\mesdocs\lecture.c  Page 2/2

    // Reserver et initialiser l'espace pour le vecteur de donnees
    vec = dvector(1,Ndata);
    for (i=1;i<=Ndata;i++) vec[i] = 0.0;

    // Lire les points de pr_point a dr_point
    i = 1;
    for(;;)
      {
      if (feof(Fd) || (i > Ndata)) break;
      *Npt = i;
      // Sauter les colonnes non utilisees
      j = 1;
      for(;;)
        {
        if (j >= colonne) break;
        fscanf(Fd,"%lf",&tmp);
        j++;
        }
      // Lire la colonne specifiee
      if (fscanf(Fd,"%lf",&vec[i]) != 1)
        {
        vec[i] = 0.0;
       *Npt = i-1;
        break;
        }
      // Lire les caracteres sur la meme ligne jusqu'a la ligne suivante
      do fscanf(Fd,"%c",&ch); while (ch != '\n' && !feof(Fd));
      i++;
      }

    // Fermer le fichier et retourner le vecteur de donnees
    if (fclose(Fd))
       printf("\nLe fichier %s ne peut etre ferme..",nom_fichier);


    svec = SousVect(vec,*Npt,1,*Npt);
    free_dvector(vec,1,Ndata);
    return (svec);
    }
```



Mar 03, 01 17:32 d:\tools\tools\declarmpeg\nrutil.h Page 1/1

```
// Fonctions d'allocation d'espaces vectoriels

float   *vector(int,int),
float   **matrix(int,int,int,int);
////////////////////////////////////////////////////////////////////////////
// Nrutil.h
////////////////////////////////////////////////////////////////////////////

float   **convert_matrix(float *,int,int,int,int);
double  *dvector(int,int);
double  **dmatrix(int,int,int,int);
int     *ivector(int,int);
int     **imatrix(int,int,int,int);
float   **submatrix(float **,int,int,int,int,int,int);
void    free_vector(float *,int,int);
void    free_dvector(double *,int,int);
void    free_ivector(int *,int,int);
void    free_matrix(float **,int,int,int,int);
void    free_dmatrix(double **,int,int,int,int);
void    free_imatrix(int **,int,int,int,int);
void    free_submatrix(float **,int,int,int,int);
void    free_convert_matrix(float **,int,int,int,int);
void    nrerror(char *);
```

Mar 03, 01 17:31 d:\mathjob\docs\annexes\nrutil.c Page 1/3

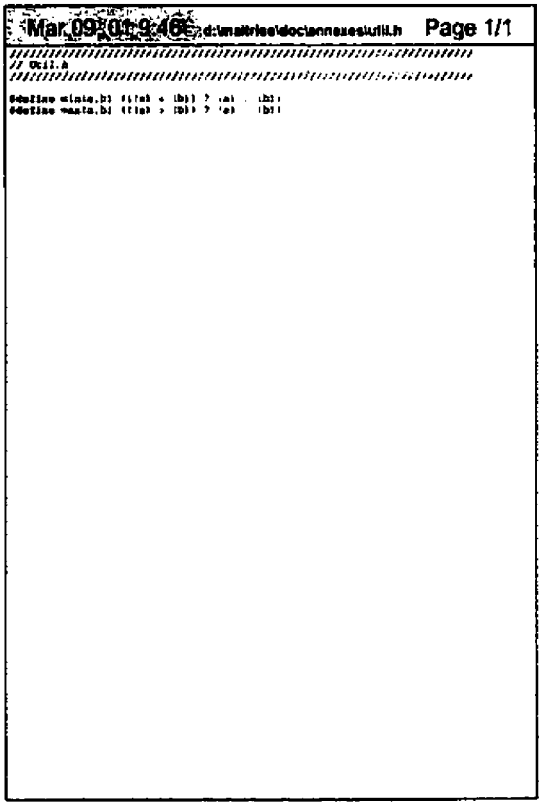
```
//////////////////////////////////////////////////////////////////////////
// Fonctions: Nrutil.cpp
//
// Date de creation:   24 avril 1996
//////////////////////////////////////////////////////////////////////////

#include <malloc.h>
#include <stdio.h>
#include "nrutil.h"

//////////////////////////////////////////////////////////////////////////
//
// ATTENTION
// ---------
// LES INDICES DE TOUS LES VECTEURS VARIENT DE 1 a N ET NON DE 0 a N-1
// AFIN D'EVITER LES CONFUSIONS DUES A L'UTILISATION DES FONCTIONS DE
// NUMERICAL RECIPES IN C
//
//////////////////////////////////////////////////////////////////////////
// Fonctions d'allocation de memoire et de traitement d'erreur
// de NUMERICAL RECIPES in C

void nrerror(char error_text[])
{
    void exit();
    printf("Erreur dans les modules de Numerical Recipes:\n");
    printf("%s\n",error_text);
}

float *vector(int nl,int nh)
{
    float *v;

    v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
    if (!v) nrerror("allocation failure in vector()");
    return v-nl;
}

int *ivector(int nl,int nh)
{
    int *v;

    v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
    if (!v) nrerror("allocation failure in ivector()");
    return v-nl;
}

double *dvector(int nl,int nh)
{
    double *v;

    v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
    if (!v) nrerror("allocation failure in dvector()");
    return v-nl;
}

float **matrix(int nrl,int nrh,int ncl,int nch)
{
    int i;
    float **m;

    m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
    if (!m) nrerror("allocation failure 1 in matrix()");
    m -= nrl;
```

Mar 03, 01 17:31 d:\mathjob\docs\annexes\nrutil.c Page 2/3

```
    for(i=nrl;i<=nrh;i++) {
        m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
        if (!m[i]) nrerror("allocation failure 2 in matrix()");
        m[i] -= ncl;
    }
    return m;
}

double **dmatrix(int nrl,int nrh,int ncl,int nch)
{
    int i;
    double **m;

    m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
    if (!m) nrerror("allocation failure 1 in dmatrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
        if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
        m[i] -= ncl;
    }
    return m;
}

int **imatrix(int nrl,int nrh,int ncl,int nch)
{
    int i,**m;

    m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
    if (!m) nrerror("allocation failure 1 in imatrix()");
    m -= nrl;

    for(i=nrl;i<=nrh;i++) {
        m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
        if (!m[i]) nrerror("allocation failure 2 in imatrix()");
        m[i] -= ncl;
    }
    return m;
}

float **submatrix(float **a,int oldrl,int oldrh,
        int oldcl,int oldch,int newrl,int newcl)
{
    int i,j;
    float **m;

    m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
    if (!m) nrerror("allocation failure in submatrix()");
    m -= newrl;

    for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

    return m;
}

void free_vector(float *v,int nl,int nh)
{
    free((char*) (v+nl));
}

void free_ivector(int *v,int nl,int nh)
{
    free((char*) (v+nl));
}
```

Mar 03, 01 17:31 d:\mathjob\docs\annexes\nrutil.c Page 3/3

```
void free_dvector(double *v,int nl,int nh)
{
    free((char*) (v+nl));
}

void free_matrix(float **m,int nrl,int nrh,int ncl,int nch)
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}

void free_dmatrix(double **m,int nrl,int nrh,int ncl,int nch)
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}

void free_imatrix(int **m,int nrl,int nrh,int ncl,int nch)
{
    int i;

    for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
    free((char*) (m+nrl));
}

void free_submatrix(float **b,int nrl,int nrh,int ncl,int nch)
{
    free((char*) (b+nrl));
}

float **convert_matrix(float *a,int nrl,int nrh,int ncl,int nch)
{
    int i,j,nrow,ncol;
    float **m;

    nrow=nrh-nrl+1;
    ncol=nch-ncl+1;
    m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
    if (!m) nrerror("allocation failure in convert_matrix()");
    m -= nrl;
    for(i=nrl,j=1;i<=nrow-1;i++,j++) m[i]=a+ncol*j-ncl;
    return m;
}

void free_convert_matrix(float **b,int nrl,int nrh,int ncl,int nch)
{
    free((char*) (b+nrl));
}
```


Mar 09 01:9:46 d:\malbrise\doc\annexes\util.h Page 1/1
// Util.h
#define min(a,b) ((a) < (b)) ? (a) : (b)
#define max(a,b) ((a) > (b)) ? (a) : (b)